University of Cape Town

SignDIn: Designing and assessing a generalisable mobile interface for SignSupport

Marshalan Reddy

RDDMAR006

Minor Dissertation presented in partial fulfilment of the requirements for the degree of

Master of Science (Information Technology)

Faculty of Science

University of Cape Town

February 2015

Supervised by Professor Edwin Blake

Declaration

I, Marshalan Reddy (RDDMAR006), hereby declare that the work on which this Dissertation is based is my original work (except where acknowledgements indicate otherwise) and that neither the whole work nor any part of it has been, is being, or is to be submitted for another degree in this or any other University. I empower the University of Cape Town to reproduce for the purpose of research either the whole or any portion of the contents in any manner whatsoever.

Signed by candidate Signature Removed

M Reddy

13 February 2015

SignDIn: Designing and assessing a generalisable mobile interface for SignSupport

Marshalan Reddy
MSc Thesis, Department of Computer Science
University of Cape Town

Abstract

SignSupport is a collaborative project between the Computer Science departments of the University of Cape Town (UCT) and the University of the Western Cape (UWC), South Africa. The intention of the software is to assist Deaf users to communicate with those who can hear in domain-specific scenarios. The penultimate version of this software is a mobile application that facilitates communication between Deaf patients and hearing pharmacists through the use of Sign Language videos stored locally on the mobile device. In this iteration, adding any new content to the mobile application necessitates redevelopment, and this is seen as a limitation to SignSupport. The architecture hinders the addition of new domains of use as well as extending the existing domains.

This Dissertation presents the development and assessment of a new mobile application and data structure, together called SignDIn, and named as an amalgamation of the words '***Sign***', ***Di****splay'* and '***In****put'.* The data structure facilitates the easy transfer of information to the mobile application in such a way as to extend its use to new domains. The mobile application parses the data structure, and presents the information held therein to the user. In this development, the Dissertation sets out to address the following:

1. How to develop a generalisable data structure that can be used in multiple contexts of Sign Language use.
2. How to test and evaluate the resulting application to ensure that parsing the data structure does not hinder performance.

The first objective of this research aims to develop a data structure in a generalised format so that it is applicable to multiple domains of use. Firstly, data structure technologies were evaluated and XML selected as the most appropriate out of three candidates (Relational Databases and JSON being the other two) with which to build the data structure. Information was collected from the International Computer Driver's Licence (ICDL) and Pharmacy domains and an XML data structure designed passing through three stages of development. The final outcome of the data structure comprises two XML types: display XMLs holding the display information in a general format of screen, video, image, capture and input; and input XMLs holding the list of input options available to users.

The second objective is to test the performance of the mobile application to ensure that parsing the XML does not slow it down. Three XML parsers were evaluated, SAX Parsing, DOM Parsing, and the XML Pull Parser. These were evaluated against the time taken to parse a screen object as well as the screen object throughput per second. The XML Pull Parser is found to be the most efficient and appropriate for use in SignDIn.

The SignDIn application design and supporting XML data structure has facilitated a means for the easy application of SignSupport to other domains of use without the necessity for reprogramming or rewriting the mobile application code. This research makes it possible to add new content to

SignSupport without building a new mobile application. By gathering and building new SASL content, SignDIn can be used to allow Deaf people to cope in more environments where the dialogue can be abstracted appropriately.

Contents

Declaration i
Abstract ii
Contents iv
List of Figures vii
List of Tables viii
Acronyms ix
Chapter 1: Introduction 1
1.1 Background 1
1.1.1 Deafness as a Cultural Denomination 2
1.1.2 Deaf Community of Cape Town (DCCT) 3
1.1.3 Previous work with DCCT 3
1.1.3.1 Looijenstein's Work 4
1.1.3.2 Mutemwa's Work 4
1.1.3.3 Chininthorn's Work 5
1.1.3.4 Motlhabi's Work 5
1.2 Motivation 6
1.3 Research Question 6
1.3.1 Generalisability: How to develop a generalisable data structure that can be applied in multiple contexts of Sign Language use. 7
1.3.2 Speed: How to test and evaluate the resulting application to ensure that parsing the data structure does not hinder performance. 7
1.4 Dissertation Outline 8
Chapter 2: Literature Review and Background 9
2.1 Sign Language on Mobile 9
2.1.1 TISSA and Relay Systems 9
2.1.2 Mobile ASL 9
2.1.3 Mobile Sign Language and Device Interaction 10
2.2 Data Structures 11
2.2.1 Relational Databases and Structured Query Language (SQL) 11
2.2.2 JavaScript Object Notation (JSON) 12
2.2.3 Extensible Markup Language (XML) 12
2.3 Conclusion 13
Chapter 3: An Evaluation of Mobile and Data Structure Technologies 14
3.1 Android as a Platform – to CH 3 14

3.2 Requirements of SignDIn 15
3.3 Evaluation 16
3.3.1 RDB and SQL suitability to SignDIn 16
3.3.2 JSON vs XML and the suitability to SignDIn 17
3.4 XML Parsers 18
3.4.1 Evaluating XML Parsers 18
3.5 Conclusion 19
Chapter 4: The Experimental Design and Methodology 20
4.1 Assumptions and Exceptions 20
4.2 Environment 20
4.2.1 The Physical Device and Development Software 20
4.2.2 The Android OS 21
4.3 Methods 21
4.3.1 Development of the XML Data Structure 21
4.3.2 Development of the XML Parser 25
4.3.3 Comparing XML Parsers for Android 25
4.4 Conclusion 26
Chapter 5: Results 27
5.1 The XML Data Structure 27
5.1.1 Storage of Assets 27
5.1.2 The Display XML documents 29
5.1.3 The Input XML Documents 31
5.1.4 An Example of SignDIn Applied to Two Different domains. 32
5.2 SignDIn and the XML Parser 34
5.3 Comparing XML Parsers for Android 37
5.3.1 Time per Screen Object 37
5.3.2 Parsed Screens Throughput 37
5.4 Discussion 38
Chapter 6: Synthesis and Conclusion 40
6.1 Key Outputs 40
6.1.1 Generalisability 40
6.1.2 Parser Efficiency 41
6.2 Evaluation of the Research Design and Methodology 41
6.2.1 Data Structure Limitations 41

6.3 Future Directions 41
6.3.1 User Testing 41
6.3.2 Other mobile platforms 42
6.4 Main Conclusions 42
Bibliography 43
Appendix A: XML v3a – Display Screens – Pharmacy patient example 45
Appendix B: XML v3b – Input Options example - diseases 48
Appendix C: Results from 1000 Cycle Parser Test 49
JDOM Parser 50
SAX Parsers 51
XML Pull Parser 52

List of Figures

Figure 1.1 SignSupport's Conceptual Design Architecture [6] 2
Figure 1.2:Creating new content for SignSupport, implemented in two stages 6
Figure 2.1: Example of JSON Code 12
Figure 2.2: Example of XML code 12
Figure 4.1: Stage 1-3 of XML Data Structure Design 24
Figure 4.2: The XMLParserTester Welcome Screen 26
Figure 5.1: How SignSupport Stores its Assets 28
Figure 5.2: Display XML of (a) a Pharmacy screen and (b) an ICDL screen 30
Figure 5.3: Gender.xml as an example of an input XML 31
Figure 5.4: The Inputs Parser - Refer also to Figure 4.1 32
Figure 5.5: The Resulting Design of the SignDIn Application 36
Figure 5.6: Parsing Speeds in average seconds per XML Document over 1000 cycles 37
Figure 5.7 Number of Screen Objects parsed in 0.2 Seconds 38
Figure C.1: Variance in time to parse a screen object in seconds for the JDOM Parser 50
Figure C.2: Variance in time to parse a screen object in seconds for the SAX Parser 51
Figure C.3: Variance in time to parse a screen object in seconds for the PULL Parser 52

List of Tables

Table 1.1 Summary of SignSupport work from 2009 to current 4
Table 1.2 Summary of objectives, required data and methodology 7
Table 3.1: Requirements of the SignDIn Data Structure 16
Table 3.2: DOM vs SAX Comparison 18
Table 4.1: Mobile Device Specification 21
Table 5.1: Parser Throughput (number of Screens parse-able in 0.2 Seconds) 38
Table C.2: Parsing Speeds in Average Seconds per XML Document 49

Acronyms

ADT	Android Development Tools
API	Application Programming Interface
ASL	American Sign Language
AVD	Android Virtual Device
BSL	British Sign Language
CSV	Comma Separated Values
DCCT	Deaf Community of Cape Town
DOM	Document Object Model
GPS	Global Positioning System
HCI	Human Computer Interaction
HTC	High-Tech Computer Corporation
HTML	Hypertext Markup Language
ICDL	International Computer Driving Licence
ICT	Information and Communication Technologies
IM	Instant Messaging
iOS	Apple's Mobile Operating System for I-Devices
JDOM	Java's Document Object Model
JSON	JavaScript Object Notation
MobileASL	Mobile American Sign Language
NGO	Non-Governmental Organisation
OS	Operating System
OSX	Apple's Desktop Operating System (Literally Operating System Ten)
PC	Personal Computer (desktop/laptop)
RSS	Really Simple Syndication
RDB	Relational Database
SASL	South African Sign Language
SAX	Simple API for XML
SDK	Software Development Kit
SMS	Short Message Service
SQL	Structured Query Language
TISSA	Telephone Interpreting Service of South Africa
UCT	University of Cape Town
URL	Uniform Resource Locator
UWC	University of The Western Cape
XHTML	Extensible Hypertext Markup Language
XML	Extensible Markup Language

Chapter 1: Introduction

1.1 Background

SignSupport is a collaborative software development project between the Computer Science departments of the University of Cape Town (UCT) and the University of the Western Cape (UWC) [1], South Africa. The intention of the software is to assist Deaf users in communicating with the hearing in domain-specific scenarios. “Domain-specific” means that the context is within a specific scenario, the benefit of which is that the application needs to accommodate a limited vocabulary and range of responses. The application does this by using pre-recorded video-clips and images stored locally on a mobile device, negating the need for network connectivity. SignSupport serves as a means of communication between Deaf people and the hearing while empowering the Deaf with the freedom to communicate in their own language in a range of environments.

According to Reagan *et al.* there are two approaches to providing assistance to Deaf people [2]. The first views Deafness as a medical condition providing solutions to assist a Deaf individual to cope as a hearing person would through the use of hearing aids or lip-reading. Reagan *et al.* refers to this as the “pathologized” approach. It aims to allow the Deaf individual to function in a hearing-centric society by providing solutions and tools to cope with the state of Deafness in much the same way as a disabled person copes with his or her disability. The alternative approach incorporates a sociocultural perspective on deafness: viewing Deaf people as individuals who are members of a cultural and linguistic minority. This approach calls for solutions that assist Deaf people to function in a dominant culture, in much the same way as a minority language or culture would [3]. This is the view that this paper, and SignSupport follow.

The earliest version of SignSupport, (circa 2009), is a computer based application and essentially a proof of a concept [4]. All iterations since then have been hard-coded to work only in specific environments and use a fixed store of information. The penultimate version, built by Motlhabi [5] works only in a pharmacy environment and is restricted to a fixed set of questions and answers (between the pharmacist and the Deaf patient), thus limiting the list of diseases and treatments that can be utilised and, in the case of the Deaf patient, limiting the information that can be collected through the questioning process. In its current format, SignSupport cannot easily accommodate any future growth, meaning that it cannot be adapted to any new environment, nor can it be extended to include more information in the current pharmaceutical domain. Completely new iterations of the software would have to be developed every time a new domain of expertise is added or an existing one extended.

This limited capability to change means that the application must be further developed to be able to accommodate other domains without extensive software development. It is possible to facilitate this through the use of an *authoring tool*. This *tool,* as it is represented in the middle layer of Figure 1.1, is a PC-based application that enables domain specialists like pharmacists or teachers to build content for SignSupport by collating sets of media containing videos, images and text in the form of instructions and questions. These media can be transferred to the mobile device in such a way as to facilitate the sequencing and filtering of the media to make it more presentable to the user. A generalisable and consistent medium of transfer as well as a speed-efficient mobile application are

prerequisites for the Authoring Tool represented as the “XML” Interface layer in Figure 1.1. This study provides a unique approach to addressing the problem of the transfer medium and the mobile application by building both a data structure and a mobile application which will render the information held in the data structure as screens which are presented to the Deaf user. With regard to Figure 1.1, this study seeks to provide a solution for the “XML” Interface and ”Mobile App” components of SignSupport. The actual development of the authoring tool falls outside the scope of this study, but some requirements for the authoring tool can be established from this paper. The remainder of this chapter will enlarge upon the background to SignSupport to provide the context for the user environment in which the mobile application can be used.

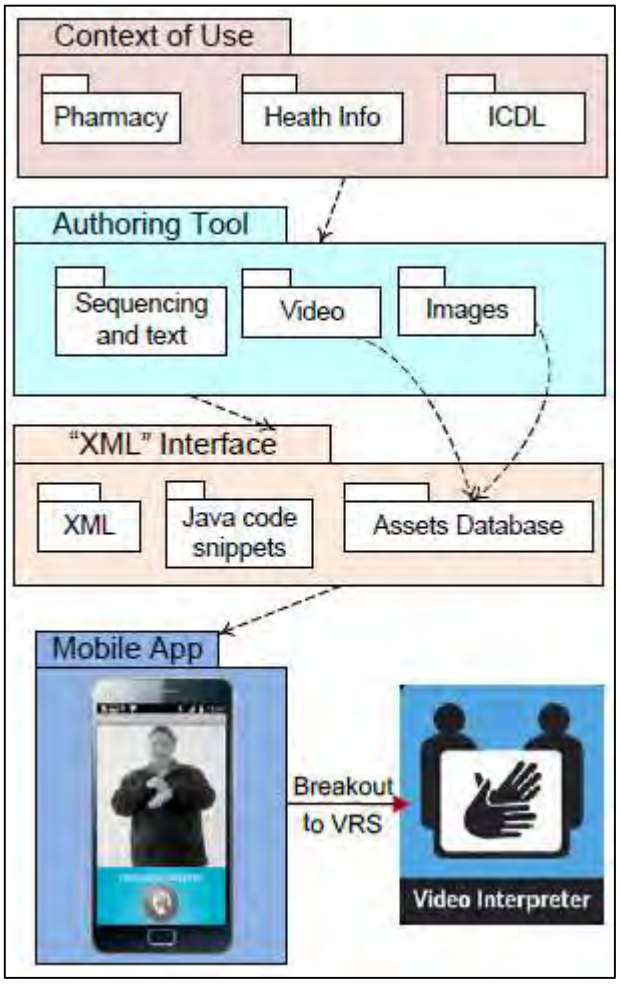

Figure 1.1 SignSupport’s Conceptual Design Architecture [6]

1.1.1 Deafness as a Cultural Denomination

From the perspective of this Dissertation and supporting research in SignSupport, the term “Deaf” with a capital D indicates a cultural denomination [7, 8].To further elaborate on this, the correct way to refer collectively to the Deaf is as “Deaf people” as opposed to simply, “the Deaf.”

Deaf South Africans have their own language namely South African Sign Language (SASL). This has not yet been recognized under the South African Constitution as one of the official South African languages. Deaf people are for the most part functionally illiterate [9, 10]. For people who can hear

and who rely on a spoken language as their first language, the ability to read and write is merely a secondary form of communication. For Deaf people, however, learning to read and write, more closely resembles learning another language in its entirety. Literacy issues amongst Deaf South Africans are expanded in Chapter 2, the Literature Review.

Further hindrances to the literacy and communication skills of Deaf South Africans can be attributed to South Africa's history of poor education for the Deaf Community. South Africa's policies of segregation caused SASL to develop differently in the segregated communities, besides a number of other causes [11]. These historical circumstances affecting the development of SASL call for the socio-cultural approach to deafness, as described earlier. Because of the variations in SASL in context and circumstance, SignSupport has been built on a foundation of fixed dialogues in relation to specific domains. These are filmed as SASL videos and then stored locally on the mobile device. The videos can be referred to as "canned" videos because they are stored locally and no dynamic change to them is possible. Using an active, mobile, SASL translation system which films a Deaf person's Sign Language and then attempts to translate it is far more difficult given the complexity of SASL, because of its inherent variations for the reasons given above. Section 2.1 of the literature review expands on the factors which affect Deaf Literacy as an explanation for the use of canned videos in SignSupport.

1.1.2 Deaf Community of Cape Town (DCCT)

The SignSupport project is being carried out in collaboration with the Deaf Community of Cape Town (DCCT), a non-governmental organisation (NGO), which not only serves Deaf people but is also almost exclusively staffed by Deaf people [12]. While Information and Communication Technology (ICT) is not DCCT's primary purpose, the NGO has a productive history of using ICT to enable and empower its members. For example, the NGO is a registered International Computer Driver's Licence (ICDL) Centre which currently offers the ICDL course to the Deaf Community. The DCCT has a long history of cooperation with UCT and UWC in the development of telecommunication aids [4, 5, 13-15]. University Researchers and Academics who have worked with DCCT still play an integral part in the progress and development of its ICT platform. Today, these researchers use DCCT to build new content for the SignSupport application, attempting to apply it to new domains, while also extending its current domains.

1.1.3 Previous work with DCCT

For almost a decade, UCT and UWC have worked closely with the DCCT to build electronic communication aids for Deaf people. [16]. The following section examines only the most recent and relevant work in line with the SignSupport application. It shows how the application evolved from PC to mobile and how the principles have been applied to different fields. The work done in previous iterations is summarised in Table 1.1 below.

Table 1.1 Summary of SignSupport work from 2009 to current

Year	Author	Title	Platform	Environment	Comments
2009[4]	Looijenstein	The design of a deaf-to-hearing communication aid for South Africa	PC	Doctor/Hospital	Proof of PC concept
2010[14]	Mutemwa	A mobile deaf-to-hearing communication aid for medical diagnosis	Symbian S60 Browser	Doctor/Hospital	Mobile Device *Also had an authoring tool on PC.
2012[17]	Chininthorn	*Communication tool design from deaf to hearing in South Africa*	Android	Pharmacy	Android Device – design focused
2013[5]	Motlhabi	Usability and content verification of a mobile tool to help a deaf person with pharmaceutical instruction	Android	Pharmacy	Hard-coded. Cannot extend/add new videos or apply to new domains.

1.1.3.1 Looijenstein's Work

The earliest version of SignSupport is a proof of a PC-based concept. The context of this version is a doctor and patient scenario and facilitates two-way communication between the doctor and the patient. The scope of this context later proved to be far too large to build structured conversations. The Deaf patient is asked questions about his/her symptoms by means of SASL videos. The answers to these questions are presented to the doctor in text format. Using a dictionary of predefined responses, the doctor can then draw up his response, which is presented to the patient in SASL.

1.1.3.2 Mutemwa's Work

Mutemwa's iteration involves adapting Looijenstein's version to run completely on a mobile device. Two Nokia Smartphones are used, the E71 and N82. The phones run the Symbian Series 60 Operating System (OS) and the application is run through the phone's browser. Because of this, Mutemwa used a combination of Adobe Flash and Extensible Hypertext Markup Language (XHTML) to present the application to the user. His application was not native to the Symbian OS and hence could not make use of all the features of the phone. At the time, the latest Android iteration was Froyo (V2.2) the use of which was not as widespread as more recent Android iterations [18]. Mutemwa also introduces a content authoring tool, a PC-based application to help build conversations and catalogue SASL videos so that different conversations can be loaded for the appropriate scenarios. This is the first introduction of an Authoring tool. Mutemwa's application can only facilitate one-way communication from the patient to doctor. This is a major functional shortcoming, and in addition to the shortcomings of the phone and the (OS), makes the application difficult to use and impractical. At this stage in the

SignSupport life cycle, it also becomes evident that a doctor's office environment is too large and difficult a context to program in the SignSupport solution. The sequences and variances in conversation require too large a database of SASL videos. The next iteration, by Chininthorn deals with some of these shortcomings, but some are still carried over to Motlhabi's 2013 iteration.

1.1.3.3 Chininthorn's Work

Chininthorn's first major contribution to extending SignSupport is its application to a pharmacy context. A pharmacy context has a far narrower scope of dialogue and it is thus easier to program this application. A pharmacist communicates a very specific and repeatable set of facts to his/her patient such as dosage amounts, frequency of dosage and side-effects. Chininthorn's background is in Industrial Design, therefore her contribution does not include the development of the software itself, but it does set a framework for the use of the software and what is required of it. This stage was important progress, as it focuses the Sign-support application on one domain that is specific enough to establish the bounds of a reasonably strict dialogue.

1.1.3.4 Motlhabi's Work

In his 2013 iteration of SignSupport, Motlhabi extended the application by developing it for the Android OS. For the first time, the application runs on a Samsung Smartphone in the native language of the OS (Java). The application can, and does make use of an Application Programming Interface (API) native to the Android OS to facilitate phone-specific features like vibration feedback when making a selection and even the use of the camera. The new phone is a major improvement to the previous one. It has enough local memory to hold a large database of videos. At the necessary resolution, Motlhabi's database of videos takes up 327 Megabytes on a phone that has approximately 25Gigabytes of local storage – which was also extendable. This set of videos provides an excellent source from which a start can be made to designing the data structure required in the first objective, which is set out in Section 1.3.1 and addressed later in Section 3.3.

At this stage, SignSupport can collect information from the Deaf user through the use of SASL videos. This iteration also involves the recording and transcribing of these videos after collecting data from the pharmacists and patients. The phone is then handed to the pharmacist, who inputs the necessary medications and dosage regime and can even include a picture of the medication (taken with the phone's camera). Finally, a Deaf patient can select the prescribed medication and view the instructions in sign language videos. Motlhabi has produced and tested these videos extensively for content verification.

Motlhabi's result is a self-contained application in that it does not require a desktop computer or even a wireless or data network connection. The major downside is, however, that it is very difficult to build any new dialogue for this application. In this iteration, it caters for a limited number of diseases and medications and in the pharmacist's domain only. Adding a new disease or medication requires the developer to create a new version of the software. Extending SignSupport to further domains involves changing the application logic at an even deeper level. Furthermore, no Authoring tool is used in this iteration.

1.2 Motivation

SignSupport is on its way to becoming a usable platform to facilitate Deaf communication. One obstacle in its development is the process involved in creating new content. The process still contains a number of challenges which need to be addressed. When any new domain is added, researchers must build a dialogue, translate it to SASL, film the SASL dialogue videos, and then verify the content. After this, the videos must be linked to their supporting content. (Stage 1 of Figure 1.2). These challenges must be addressed by a Desktop-based Authoring Tool. The deployment of all this content to a mobile device remains a further challenge. Once the dialogue is built and videos filmed, the mobile application must accept the information and a usable interface created for the Deaf user to be able to consume the content. A channel to facilitate the transfer of this information from Authoring Tool to mobile device must also be developed (Stage 2 of Figure 1.2).

Future researchers and content contributors will be able to expedite the deployment of content if the mobile application is built in such a way that the transfer of data is a simple and repeatable process. This means building a mobile application that can be extended merely by transferring new content to the local storage on the mobile device. With the right content, the application can be adapted and enlarged for use in a variety of ways. If the concern for mobile data usage can be set aside or overcome, the application can even be extended to collect this information from an online resource. This Dissertation aims to address this gap in the current versions of SignSupport.

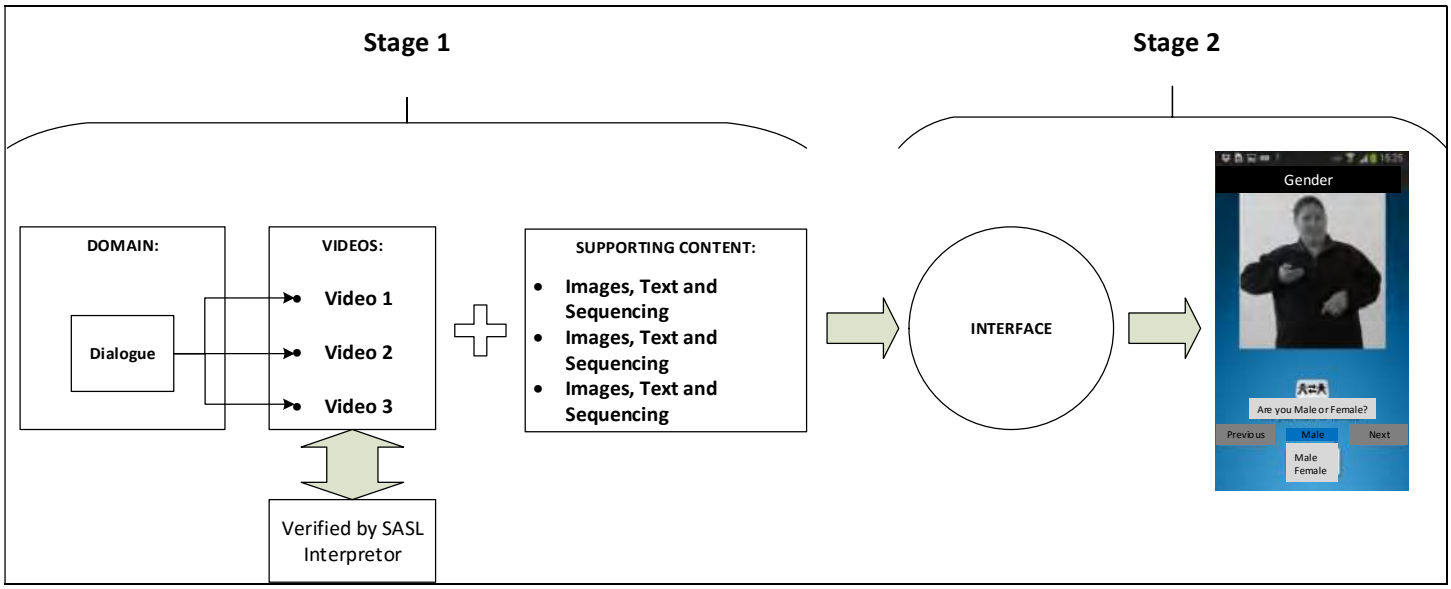

Figure 1.2:Creating new content for SignSupport, implemented in two stages

1.3 Research Question

The overall aim of this Dissertation is to build on previous iterations of SignSupport by building an application to act as a data transfer channel between the PC based Authoring Tool and the mobile user. The application will accept information outputted by the Authoring Tool and display it in a logical and usable sequence to the Deaf user. If and when necessary, the application will prompt for and collect user inputs. In short, this mechanism will facilitate the **Di**splay of information to a user and **In**put of user information. It will henceforth be referred to as **SignDIn** and named as an amalgamation of the words '***Sign***', ***Di****splay'* and '***In****put'*. Any future reference to SignSupport therefore refers to the

entire suite of applications, or the entire SignSupport project. This research aims to address the following question:

Can a display and input mechanism which is both generalisable and fast be built for SignSupport?

This question can be broken down into two primary objectives:

1.3.1 Generalisability: How to develop a generalisable data structure that can be applied in multiple contexts of Sign Language use.

1.3.2 Speed: How to test and evaluate the resulting application to ensure that parsing the data structure does not hinder performance.

To achieve the first objective, this Dissertation describes the process necessary to build a generalisable data structure in a medium or language that can be transferred between a PC and mobile device. In the context of this study, to "generalise" the data structure means to build the data structure in a general format so that it is broadly applicable to new domains. The data structure can therefore hold the general contents of the information from any domain (videos and supporting content) and also facilitate the display and sequencing of the content. The data structure holds the contents passed from the Authoring Tool to the mobile device. This process is illustrated in Figure 1.2. Stage 1 of Figure 2 falls under the realm of the Authoring tool and its user, the domain specialist. Stage 2 will be fulfilled with the data structure and the mobile application, together called SignDIn.

To achieve the final objective, a mobile application is built to parse the data structure and display its contents in a format that does not hinder responsiveness to any great extent. Various types of parsers are evaluated for their responsiveness, and the most appropriate one used in the final product.

This research will create the foundation for future researchers to extend SignSupport because it will establish a systematic, standardised means of transferring data from the Authoring tool to the mobile application. The simple, generalised data structure this Dissertation proposes will facilitate the addition of new domains and SASL videos to SignSupport, should subsequent researchers wish to do so, without having to redesign the logic of the mobile application. Table 1.2 gives a conceptual overview of this Dissertation, along with the data required to achieve each objective and the methodology followed to acquire it.

Table 1.2 Summary of objectives, required data and methodology

Objective	Required Data	Methodology
1. Generalisability	A data structure capable of holding content from multiple domains, built with an appropriate technology	• Justify the technology to be used • Design the data structure
2. Speed	An application tested against appropriate measurement criteria	• Collate and assess data.

1.4 Dissertation Outline

This chapter provides the context and background for the SignDIn application.

In **Chapter 2** the supporting literature is analysed critically to give a general idea of the state of Sign Language translation, then other attempts at mobile translation are examined to compare and assess their shortcomings and ascertain whether there is any opportunity for enhancement and extension. Chapter 2 also introduces three potential technologies for use as a data structure.

In **Chapter 3** important preliminary information is provided before approaching the two main objectives of this paper, namely, the evaluation of three suitable technologies for data structures and the justification of one for this Dissertation. The goal here is to facilitate the achievement of the first objective, by explaining why the chosen technology was deemed the most appropriate for the SignDIn data structure. The Chapter gives a detailed description of the requirements for the data structure, then evaluates each technology against these requirements. Finally, parsers specific to the chosen technology are introduced and their significance in achieving the second objective is explained.

In **Chapter 4** the data structure above is implemented for the development of an experimental design for the purposes of achieving the research objectives. The chapter covers the design, development and use of a mobile application utilising the data structures and related data and demonstrating how the objectives are achieved. This chapter also describes the prevailing exceptions and assumptions, and the environment in which the design is carried out. The methods used to solve the objectives of SignDIn are fully elaborated.

In **Chapter 5** the results of the previous chapter are discussed. Here, the reader gains a deeper understanding of what exactly the SignDIn application does and how it fits into SignSupport.

In **Chapter 6** the Dissertation is concluded and insight is given into the limitations of the work and the potential for future research and development.

Chapter 2: Literature Review and Background

This chapter gives an overview of Sign Language translation on Mobile. Firstly, a brief coverage of Sign Language is given and an explanation why translation is done in this particular way. Section 2.1 looks at other attempts at Sign Language translation. Section 2.2 introduces data structures, looking at Relational Databases, (RDBs) JavaScript Object Notation and Extensible Markup Language. And finally, Section 2.3 concludes this chapter.

The obvious question raised when considering sign language translation is why it is even necessary for computer aided Sign-Language translation when a Deaf user could simply use text to text communication as in the case of a Short Messaging Service (SMS), Instant Messaging (IM) or email?

In his research, Stokoe [19] outlines the visual communication systems of American Sign Language (ASL) and shows that sign language is not universally identical. It is as unique in different countries or communities as any native languages and cultures are. There are subtle intricacies (like gestures) that at the very least must be communicated by video or audio. Furthermore, he explains that sign language can develop in parallel to a community's culture and language. This explains why American Sign Language (ASL) and British Sign Language (BSL) can and does differ to a large extent from South African Sign Language (SASL). The cultural inputs and experiences of the Deaf person have shaped their specific language. Deaf readers may have more trouble comprehending the written word than do hearing readers [20]. The phonological reading theory provides an explanation for this, associating reading skills with phonological abilities. Basically the ability to hear or sound out a word facilitates reading that word. This significantly affects the reading ability of those with prelingual deafness, such as those Deaf from birth. Other factors such as the context or emotion may be lost in text or SMS. This also explains why it is so difficult to create software that can translate Sign Language actively. The subtleties in expression and wide variety of context-specific definitions make this far too complex a task.

2.1 Sign Language on Mobile

2.1.1 TISSA and Relay Systems

A common approach in developed countries is to use a relay system to provide non-textual mobile communication service for Deaf users. The Telephone Interpreting Service for South Africa, or TISSA was intended to be such a system. The service was intended to provide access to government services in SASL. A SASL interpreter would be contactable by telephone or videophone and act as a bridge between the Deaf person and the hearing party. The pilot study for the system was carried out in 2002, but no successful implementations resulted. This is probably due firstly to the cost of implementation, and use, and secondly to the availability of landline phone systems over mobile [21-23].

2.1.2 Mobile ASL

In 2006, in the United States, Cavender *et al.* [24] recognise that access to the American mobile network for Deaf people is largely limited to text messaging (SMS). This forces the Deaf to Deaf communication to be conducted in written English as opposed to American Sign Language (ASL) [24]. This early work suggests that mobile devices should be used to transmit video calls to assist Deaf people. Today there are systems able to facilitate this, e.g. Apple's Facetime (exclusive to Apple iOS

and Apple OSX devices) and Microsoft's Skype (almost platform independent) to name two. What Cavender *et al.* foresee in 2006, is, however, hindered by the prevailing technology. 3G networks were not ubiquitous enough at the time to carry the transmission of video calls; neither were many Smartphones equipped with front-facing cameras. Another shortcoming of the system Cavender evaluates is that it is necessary to place the phone in front of the user on a stand of some sort to hold it. Many of the problems foreseen in this study are later resolved by mobile tablets, more advanced Smartphones, and faster and more widespread mobile Internet connections. However, Cavender's study does not resolve the issue of how Sign Language can be translated to the spoken word and vice versa, which is resolved by SignSupport, thereby facilitating communication between Deaf people and those that can hear.

In 2011, Kim *et al*. evaluate a software program, which enables sign language video on mobile devices over conventional cellular networks in the United States [25]. By 2011, the technological deficiencies in Cavender's work, to a certain extent, had been overcome, but the software (MobileASL) did not fully overcome newer shortcomings. The experiment participants are given a HTC Tytn II Smartphone with the MobileASL application preloaded onto it. The phone itself is partly to blame for some of the shortcomings. Users complain in specific that the battery life is drastically shortened when using MobileASL. This is largely because all the processing required to record, transmit, and play the video back is carried out by the phone. SignSupport on Android overcomes this because the processing is carried out on the SignSupport back-end. To quote the text:

> "*Participants also pointed out that the phone was too big and the tilt of the screen needed to go further. They also mentioned wanting a touch-screen interface more easily controlled by a fingertip than a stylus." [25]*

Later model Smartphone devices meet these demands by offering an extended battery life, better touch screen interfaces and an overall improved HCI design. It should also be noted that MobileASL does not provide any translation mechanism, but only a Deaf to Deaf communication tool.

2.1.3 Mobile Sign Language and Device Interaction

In 2010, Jayant *et al* carry out a study demonstrating that V-Braille can be used by Deaf people and those who are both deaf and blind to read individual characters and sentences on mobile devices [26]. The methodology facilitates a touch screen divided equally into six parts, to mimic the six dots in a single braille cell. This is a unique example of Human Computer Interaction (HCI), although it is not universally applicable to Deaf users. The users know which cell they are using and which Braille icon they are typing by means of vibration feedback. This gives an interesting take on HCI for the sensory-impaired. No other literature in the review covers the use of touch as a method of feedback. The study shows that V-Braille has the potential for use by both deaf and blind people to interact with the world through mobile technology as an alternative method of communication.

In a similar approach, Azenkot *et al* develop GoBraille, two related Braille-based applications that provide feedback on public transport systems [27]. Go-Braille was codesigned by a deaf-blind person which resulted in a minimalist interface, with short input and output messages. The application works in two parts, firstly to provide real-time bus arrival information and secondly crowd-sourced information about bus stop landmarks. The key to this approach is that the crowd-sourced information is provided by GoBraille users, who are themselves sensory-impaired, and therefore provide only the relevant information needed to help other users. The main drawback of this approach is that it

requires the addition of a Wi-Fi-enabled Braille display. The work is important to this research, as it shows how important simple interfaces are for Deaf people.

A study by Jones and Johnson [28] points out that even fully capable (non-sensory-impaired) users can only focus on a certain number of tasks at one time. Mobile users, in specific, want to carry out atomized tasks as quickly and efficiently as possible. The study proposes that the design should focus on specific, functional tasks. This supports the view of much of the other literature in this section; i.e. that simple and user-friendly design should be at the core of the Android SignSupport application. This is even more valid when designing for the sensory-impaired.

The literature covered thus far shows the failures of attempting to implement relay systems in developing countries. The technological setbacks experienced in MobileASL are accommodated in the newer Samsung Galaxy SIII. Also of importance, is designing a simple, standardised interface. What follows provides a general overview of the history of SignSupport and its current shortcomings and scope for growth.

2.2 Data Structures

What SignSupport lacks is the ability to be refreshed or updated as new requirements are set. This can be done with the use of a data structure, which is a particular way of storing data so that it can be easily and efficiently used [29]. As part of this experiment, three technologies are evaluated for suitability as a data structure, against the benchmark requirements of SignDIn.

Chapter 3 evaluates the three established technologies in the field, Relational Databases and SQL; JavaScript Object Notation (JSON); and Extensible Markup Language (XML). The following paragraphs are an introduction to these topics.

Relational Databases and Structured Query Language (SQL)

A Relational Database (RDB) is one where information is stored about the data as well as how the data relates to other data. All data relationships are represented as two dimensional tables [30].

A Relational Database Management System (RDMS) is essentially software that controls the reading, writing and modification of the Relational Database. The RDMS achieves this through the use of Structured Query Language, or SQL.

Relational Databases comprise collections of related tables with each table holding information in rows and columns. Each row equates to an entry in the database and can be linked to information in another table. Much of the purpose of Relational Databases is to allow for connections and correlations to be built between large and complex sources of information.

JavaScript Object Notation (JSON)

```
{
      "Root": {
            "child1": {
                  "tag1":"text",
                  "tag2":"text"
            }
      }
}
```

Figure 2.1: Example of JSON Code

Figure 2.1 above shows a sample of JSON text containing the same data as the XML document in Figure 2.2. JSON is an open-standard format using human-readable text to transmit data objects in attribute value pairs. JSON is used to transmit data over the Internet, in a server-client scenario. As in the case of XML, JSON is language independent. It is a popular alternative to XML, because it offers benefits in performance when used in a client/server scenario, i.e. where transmission is over a network interface [31]. Although this is not applicable to SignDIn, it should be noted that there is no formalised schema system for JSON, as there is in the case of XML.

Extensible Markup Language (XML)

XML Stands for Extensible Markup language and is a text-based markup language derived from SGML (Standard Generalized Markup Language). The XML document is used to display and organise data. XML tags in the document can be used to identify individual elements of data. XML does not qualify as a programming language because it does not instruct a computer to perform any task. It is usually stored as simple text files and processed by software capable of interpreting (or parsing) XML.

Figure 2.3 shows a simple XML document. For the purposes of SignDln, this describes all the necessary characteristics of an XML document. Line one is standard in most XML documents. This is the XML declaration which declares to any parser or compiler, that this is an XML document. Line 3 starts with the first and only element of this document, called the Root. The Root element ends on line 8 and contains an element called child 1. Child 1 has two further child elements called Tag1 and tag2, each containing a string called “text”. This XML document can be said to have one element, and go down to a depth of 3 levels. The document can be described as being well-formed, because the tags nest within tags, and do not overlap. No Document Type Definition (DTD) is applicable to this document, neither is there one applicable to the XMLs used by SignDIn.

```
<?xml version="1.0" encoding="UTF-8"?>

<Root>
      <child1>
            <tag1>text</tag1>
            <tag2>text</tag2>
      </child1>
</Root>
```

Figure 2.2: Example of XML code

XML was developed by an XML Working Group under the World Wide Web Consortium in 1996. The design goals of XML provide a strong case for implementing it in a project such as SignDIn.

The goals, taken from www.w3.org are outlined below:

1. XML shall be straightforwardly usable over the Internet.
2. XML shall support a wide variety of applications.
3. XML shall be compatible with SGML.
4. It shall be easy to write programs, which process XML documents.
5. The number of optional features in XML is to be kept to the absolute minimum, ideally zero.
6. XML documents should be human-legible and reasonably clear.
7. The XML design should be prepared quickly.
8. The design of XML shall be formal and concise.
9. XML documents shall be easy to create.
10. Terseness in XML markup is of minimal importance.

This has resulted in a markup language with three important characteristics making it useful in a variety of systems and solutions.

1. XML is extensible: XML essentially allows you to create your own language, or tags, that suit your application.
2. XML separates the data from the presentation: XML allows you to store content without regard to how it will be presented.
3. XML is a public standard: It is available as an open standard.

The following section concludes the literature review.

2.3 Conclusion

The literature has shown sufficiently, the current state of the discipline into which SignDIn fits. The reasons for using domain-specific dialogue and the resulting canned videos are justified. Android is demonstrated to be the platform of choice when developing this software. There has also been sufficient coverage of a number of other approaches to using mobile communication to facilitate conversation between Deaf people and the hearing. TISSA is merely one example of an unaccomplished relay system in developing countries.

Examples such as MobileASL and GoBraille show how difficult pre-SignSupport attempts at Deaf-mobile usage proved to be. MobileASL also establishes why it is more practical to store videos locally on a mobile device rather than stream them over the Internet or have them sent to the phone.

Three options for data structures are introduced, namely Relational Databases, JSON and XML with brief descriptions of how they work.

Before attempting to fulfil the Dissertation objectives, it is necessary to evaluate Relational Databases, JSON and XML, fully, as potential solutions to a data structure for SignDIn. Chapter 3 also justifies the use of this technology in delivering these requirements to the SignDln application. Chapter 4 looks at fulfilling the objectives of this Dissertation.

Chapter 3: An Evaluation of Mobile and Data Structure Technologies

Chapters 3 and 4 cover the strategies used to fulfil the research objectives. Before approaching the two objectives (dealt with in Chapter 4), Chapter 3 examines the reasons why Android is the chosen platform and explains how the most appropriate technology is selected for the data structure. Alternatives are evaluated and the evaluation criteria explained. This chapter also introduces the main types of parsers used to satisfy the final objective of this Dissertation. In short, the chapter gives preliminary answers to the first research question, and introduces the parsers to be used in satisfying the second research question.

Section 3.1 of this chapter looks at why Google's Android is the platform of choice for the SignDIn system. Section 3.2 of this chapter commences with an explanation of the main functionality that the data structure fulfils. These requirements are abstracted from the research questions. In Sections 3.3 to 3.4, the three data structures, Relational Databases (RDB), JSON, and XML are evaluated according to a set of priorities. These technologies are introduced in Section 2.2 of the literature review. In Section 3.3, each technology is evaluated against the SignDIn requirements to establish the most appropriate one for providing the data structure. After establishing the technology of choice, chapter three introduces the XML parsers to be evaluated later in Chapter 4.

3.1 Android as a Platform – to CH 3

The 2011 paper, "*Smart Smartphone Development: iOS versus Android*" [32] attempts to establish which of the two operating systems: Apple's iOS, or Google's Android is the most appropriate for teaching the development of mobile applications. This research indicates that the Android mobile operating system is the most practical software platform for teaching and that it is also the best operating system for use in research.

The writers establish first, that the hardware requirements for the respective Software Development Kits (SDK) favour the development on Android as the Android SDK (known as Eclipse) is compatible with the Microsoft, Apple, and Linux operating systems. On the other hand, iOS SDK (known as XCode) is only compatible with the Apple Mac operating system from the OSX 10.6 Snow Leopard version onwards, and then only on Apple Mac computers with an Intel processor (these are the later and sometimes more expensive iterations).

Secondly, the paper establishes that at some stage, iOS development requires an iDevice (iPad, iPhone or iPod Touch) for full usability testing. This is because the simulator included in the XCode SDK does not include the complete simulation of features like a Global Positioning System (GPS), the accelerometer, compass or camera. In contrast with this, the Android SDK provides a comprehensive desktop simulator which allows for creating features on Android Virtual Devices (AVDs) which can be instantiated with different screen resolutions and memory sizes while also being subjected to GPS readings and other inputs.

In terms of programming language, iOS applications are written in Objective-C. An object-oriented language based on C. Android models and controllers are written in Java, but Scripting Language for

Android (SL4A) provides the access to the Android API through Python and other languages. Both Eclipse and XCode adopt the Model-View-Controller methodology.

Both operating systems have a large number of on-line resources: instructional videos and books including books to assist professional developers.

In terms of cost, Apple's model of selling the operating system and hardware as a single product has closed the market for competitors wishing to provide cheaper iOS platforms. In most cases, this makes iOS devices more expensive than the Android devices. On the other hand, Google provides its Android OS as an open-source product. This, again, opens up the market to allow manufacturers to provide a variety of handsets - all running the Android operating system. This means that while it is possible to buy an Android device at much less cost than an iOS device, this is not necessarily a guarantee of Apple comparable features and quality. However, it does provide the same operating system at a lower cost.

The approach to the penultimate SignSupport version seems to align with Goadrich's findings [32] as researchers choose to develop their Sign Language applications for the Android Operating System. Goadrich's work puts forward a compelling argument for the use of Android rather than iOS, especially in a University or research environment.

3.2 Requirements of SignDIn

Table 3.1 provides a summary of the requirements for the SignDIn data structure. There are 11 requirements sorted in a descending order of priority, and in an ascending order of importance within each priority. The Priority rating is classified as follows:

- **Priority A:** The feature is necessary for the functioning of the data structure.
- **Priority B:** The feature can be considered desirable and beneficial for the developers and researchers working on SignDIn.

In order to meet Priority A, the ability to hold data in plain text is necessary because SignDIn holds information that is generalisable to all the domains of SignSupport. Storing this information in plain text makes it far easier for the parser to handle or utilise the data. It also makes the development of the parser easier. This aligns with the first objective of this Dissertation, which is to make the data structure generalisable. In this context, future proofing means ensuring that the data structure is able to accommodate new information for the foreseeable future. Compatibility with native Android frameworks is necessary, as SignSupport is currently only available on Android devices. This is justified and explained in Section 3.1. Readability, easy adaptability, and the prevalence of the data structure lowers any barriers of entry to future developers who may wish to work with SignDIn. This is a benefit for any future undergraduate or post-graduate students wishing to work with the system, who would otherwise be hindered by its complexity.

To elaborate on the Priority B items, the ability to test different parsers easily fulfils the second objective of this Dissertation, which is to evaluate the final mobile application for the speed of screen changes. The ability to test the alternative methods in accessing the data, in turn, allows the best data reading or parsing method to be selected, and the less suitable alternatives to be discarded. Priority

B items are not necessary for the functioning of SignDIn, but are considered favourable because they will assist developers and researchers who may work on this in the future.

Table 3.1: Requirements of the SignDIn Data Structure

Priority	Item Number	Description
A	I	Able to hold Data in the form of strings in a generalisable format
	II	Provides a degree of future proofing - Extensible to other domains - Extensible to other mediums - Generally universal
	III	Compatible with the Native Android Framework
	IV	Easy to read and, if necessary to be built manually
	V	Easy to build applications that use the information in the data structure
	VI	Universal enough to be easily accepted in the mobile development community
	VII	Able to provide all the features in one solution.
	VIII	Must be freely available and widespread enough to source information easily.
B	IX	Easy and simple to test different software parsers/readers
	X	No APIs or third party plug-ins for these parsers
	XI	Software to work on and build the data structure must be freely available in complete forms

3.3 Evaluation

This section evaluates the three technologies and provides the necessary information as an answer to the first research objective, of generalisability (Section 1.3.1).

The rest of this section evaluates the performance of each technology in meeting the requirements outlined in Table 3.1. Section 3.3.1 discusses the Relational Database, and Section 3.2 discusses JSON versus XML.

3.3.1 RDB and SQL suitability to SignDIn

Referring to the Priority A items from Table 3.1, Relational Databases are able to hold information in the forms of strings, as required by SignDIn. RDBs can also lock content types to hold only information in strings of certain formats or lengths. RDBs provide a degree of future proofing, as they are prevalent in modern day databases and data structures. The native Android Operating System includes frameworks for working with SQL databases, which are stored locally on the phone. This is a benefit for SignDIn, as it means that no data transfer over a network connection is necessary. However, Relational Databases may not meet SignDIn's requirements for readability and the ability to be built manually. A Relational Database is necessary, when large amounts of information need to be accessed and worked with. Large databases cannot be manually populated through a process entailing content building for SignSupport. SignSupport's video and content sources are not yet large enough to justify the use of RDBs. Also, the SignSupport data does not change frequently enough to justify this. In terms of the ease of building applications using SQL, this is not an impossible task, as Android already

contains the necessary framework, and all the developer needs to do is build the necessary knowledge to work with the database actively.

SQL does not need testing with different types of plug-ins or parsers, (Priority B) as it is typically used for Relational Databases. Testing its efficiency falls outside the scope of this work and will not necessarily be of benefit to SignDIn. There are many software options such as MySQL which work directly with SQL and provide a high level of functionality free of charge.

Relational Databases offer the ability to rapidly modify data in large quantities, as well as the ability to manipulate data. Overall, however, RDBs and the reliant SQL do not fulfil the requirements of the SignDIn data structure sufficiently, as the scoring in Table 3.2 reflects.

3.3.2 JSON vs XML and the suitability to SignDIn

JSON is an elegant solution to many of the challenges faced in designing a data structure. The major shortcoming of JSON, however, is that it is less readable than XML. XML contains a simple tag and content design, while JSON makes use of nested brackets and an attribute value syntax. JSON also does not use a Schema, so considering that SignDIn's content will be manually built in the early stages, it would make troubleshooting difficult. JSON's major improvement over XML should also be noted. It is smaller and thus more efficient when used in a client-server relationship. XML's verbosity is largely due to the use of tag names to open and close every element that is not empty. The duplication of the opening and closing tag results in a higher utilisation of bandwidth and processing power.

XML is less efficient than JSON, but efficiency in bandwidth usage and processing requirements has not been identified as a major requirement for SignDIn. There are three core reasons for this, firstly, all videos are stored locally, (no bandwidth usage), and secondly, the data structure only needs to be created or updated when new material is required. As explained in Motlhabi's work [5], this is not a simple process, neither is it a frequent occurrence. Lastly, if there is a limit to the processing power, it is reasonable to expect that this limit will be reached when the SignDIn videos are played, rather than when the data structure is parsed. In certain circumstances, XML does offer benefits over JSON. Firstly, XML allows the user to validate whether an XML document is correct before transmission. This can be included as a requirement in the Authoring Tool, to ensure the XML being transferred has been validated. Secondly, XML can hold rich data types, like dates, date time, etc., while JSON can hold only strings, numbers, Booleans or arrays. Currently, this is not required by SignDIn, but XML does allow for it, should the need arise. XML is an easier data structure to read and build manually, because of its close resemblance to HTML and its clear, simple use of a tag and value syntax. Considering that most of the developers working on SignSupport are graduate students who may not have professional experience with data structures, XML provides a simple introduction to this. There is one type of JSON parser for Android, because parsing JSON is much simpler than parsing XML. This is not ideally suitable for SignDIn, as it offers no alternatives when the speed of the parser is being assessed. Any assessment of speed is actually an assessment of the code implementing the parser, rather than the parser itself and this does not offer any benefit for future developers implementing this parser on other platforms.

To summarise, XML has been chosen as the data structure of choice because it of its easy readability, and because it allows for the evaluation of multiple parser technologies.

3.4 XML Parsers

The second objective of this Dissertation is to assess the speed between the screen changes in the SignDIn mobile application. To a large extent, this involves assessing the speed at which the application parses the XML. The next section introduces the different types of XML parsers.

3.4.1 Evaluating XML Parsers

When software reads an XML document, it is described as parsing. There are essentially two differentiated parsing methods for XML: SAX (Simple API for XML) parsing [33] and DOM (Document Object Model) parsing [34]. Table 3.2 briefly describes the core differences between SAX and DOM Parsing. The Android mobile operating system natively supports both parsing methods, and can also include third party parsers, like JDOM, (Java Document Object Model). Third party parsers offer benefits such as being easier to code, improved readability or better compatibility with the native software. They may vary widely depending on their implementation, which can influence the speed between implementations, but not the speed between different types of parsers. Regardless, it should be noted that this Dissertation does not assess the parsers or their suitability for SignDIn or the Android framework, but simply assesses them in terms of their speed relative to one another as outlined in the second research objective.

Table 3.2: DOM vs SAX Comparison

	DOM	**SAX**
Type of Parsing	Object-based	Event-based
Object Model	Created Automatically	Must be actively created
Sequencing	Able to sequence elements	Cannot sequence elements
Memory use	Higher	Lower
Speed of parsing	Slower	Faster
Ability to update XML	Yes	No

3.4.1.1 SAX Parser

The key characteristic of SAX Parsing is that it is event driven. It looks for relevant information in the XML document and processes it on a per-item basis. As such, SAX Parsing can be less memory-intensive than DOM parsing. Since it is not storing any data while it is scanning, it can parse an entire XML document faster than a DOM parser.

3.4.1.2 DOM Parser

The DOM parser uses object-orientated methodologies for working with information once it is parsed from the XML. It allows for the representation and manipulation of elements within the XML document as if they were objects [34].

3.4.1.3 JDOM Parser

The JDOM parser is an example of a third party parser that can easily be included in the Android SDK. It makes use of the Document Object Model, but uses a different set of classes to handle the data. It is built specifically for Java so it is easier for a Java developer to write and understand than other parsers.

3.4.1.4 XML Pull Parser

The XML pull parser is built into the Android framework. Pull parsing treats the XML document as an iterative object. It can step logically through the XML and automatically extract data such as tags and values iteratively. For this reason, a pull parser is easier to maintain than a SAX parser.

3.5 Conclusion

This chapter discusses the data structure functionality required by the SignDIn application. All SignDIn requirements have been abstracted from the research objectives, then explained and categorised into two sets of priorities. Based on the evaluation of the two sets of priority requirements, and a discussion of XML versus JSON as a suitable data structure, XML is deemed to be the most suitable technology to be used in building the SignDIn application. Its features: readability, ease of use, and lack of reliance on other software make it most appropriate for the SignDIn application.

The objective of this chapter is to provide the necessary information required to address the first research objective (Section 1.3.1). Before testing for generalisability, the technology to be used in developing the data structure had to be established. All references to the data structure from this point onwards imply that it is based on XML.

The following chapter covers the core methodology and design of the experiment used to address the two objectives of this Dissertation. Chapter 4 builds on Chapter 3 by explaining how the XML data structure is populated to provide generalisability and how the mobile front-end of SignDIn is assessed for speed.

Chapter 4: The Experimental Design and Methodology

This chapter covers the strategies used to fulfil the research objectives. In specific, it examines the development of the Android mobile application aspect of SignDIn. This is the feature of SignDIn responsible for parsing and rendering the information held in the data structure designed in Chapter 3. First, an explanation is given of the general assumptions and exceptions in the experiment. Section 4.2 provides an understanding of the environment in which the design of the software takes place and the methodologies used. Section 4.4 examines the individual steps in the methodology used to answer each research question along with describing the software designed to resolve the problems.

4.1 Assumptions and Exceptions

The following section describes the scope of the application and what is not covered or tested.

Firstly, the SASL videos are prerecorded and supplied by other, supporting UCT/DCCT projects, such as Motlhabi's work. [5]. It is beyond the scope of this experiment to establish the required dialogues used in the specific environments, or to film SASL videos relevant to these dialogues. Nor is it relevant to test the videos for correctness of content.

Secondly, the mobile application is designed for use almost exclusively by Deaf people. This gives rise to a unique set of usability requirements. It is out of the scope of this project to assess the usability of the graphical user interface by Deaf people.

Lastly, the Authoring Tool discussed earlier (See Section 1.1) is not an objective of this Dissertation. This paper does involve the development and assessment of the data structure, and the Android mobile application that parses the data structure and displays it to the user.

4.2 Environment

4.2.1 The Physical Device and Development Software

The following section describes the development environment used in designing and developing the Android application and XML data structure.

The XML parser was designed and tested on a Samsung Galaxy SIII mobile device. The specifications are included in Table 4.1. No virtual devices were used for testing, only the physical device.

The XML parser was developed using the Android Development Tools or ADT bundle (Version 0.4.2) for Apple OSX Mavericks (Version 10.9) and Microsoft Windows 7. The ADT bundle includes Eclipse (Version 4.3). The programming language used was Java (Version 1.6).

The development of the XML structure was carried out with Sublime Text (Version 2.0.2): a simple text editor. At this stage, the XML data fed into the application was put together manually. In future, it is anticipated that the Authoring Tool will automatically build the XML documents and transfer them to the application.

Table 4.1: Mobile Device Specification

Device	Samsung Galaxy SIII
Operating System	Android V4.3 (JellyBean)
Internal Storage (Gigabytes)	24
External Storage	Up to 64 GB with SD card, not used in experiment.
CPU	Quad core 1.4Ghz
RAM	1GB
Display Size	4.8” or 720x1280pixels

4.2.2 The Android OS

In the development of applications for the Android OS, each view, or screen viewed by the user is called an Activity. The first Activity starting the application is called the Main Activity. An activity displaying a list of items is a ListActivity. The screen activity used in SignDIn is a FramelayoutActivity. Activities have what is termed a “lifecycle”, where the Activity transitions between different states. At each of these transitions, the activity makes use of certain methods of the Activity Class, as listed below, in the order in which they would typically occur:

- `onCreate()` - called when the activity is created(on application start-up)
- `onStart()` – called when the activity is made visible to the user.
- `onResume()` – called when the activity is in the foreground and the user is interacting with it.
- `onPause()` – called when the activity is partially obscured by another activity and cannot receive inputs or execute code.
- `onStop()` – called when the activity is completely hidden from the user and considered to be running in the background.
- `onDestroy` – called when the activity is destroyed

The developer can align these methods with certain functions in the application. The XML parser in SignDIn runs inside the `onCreate` method for its parent activity. This is why testing the parser for speed is necessary (as in objective 3, Section 1.3.3), as a sluggish parser presents itself to the user as a momentary pause when the user is shifting between screens on the mobile device.

4.3 Methods

On commencement of this project, the architecture of the SignSupport suite of applications is as described in Chapter 1, Figure 1.1. Context of Use implies the different domains in which SignSupport can be applied. The Authoring Tool is the PC based application that allows a super user or domain specialist the ability to build content for the mobile application. The XML Interface is the medium through which information is passed to the mobile application.

The following sections of the methodology will deal with how the XML data structure is designed, and how the mobile application is designed to parse and display information from the XML.

4.3.1 Development of the XML Data Structure

This section presents the methodology applied to address the first objective of generalisability in SignDIn through the design of an appropriate XML data structure. The development of the XML data

structure can be broken down into three stages (Figure 4.1) with the end stage producing the final version used in SignDIn. In the first stage, all the information must be collected from all the current domains served by SignSupport. In the second stage, the information is differentiated into Display and Input categories in order to separate out the incompatible formats and to reduce or remove redundancy altogether if possible. The goal of the third and final stages is to achieve a means to link the display and input categories dynamically.

4.3.1.1 Stage 1

The first iteration of the XML data structure involves the input of each aspect of the SignSupport suite. This involves collecting data from different sources for each domain of SignSupport and establishing a broad set of requirements for the XML data structure. The inputs are from the pharmacy and ICDL domains. Research into building the ICDL domain ran in parallel but largely independent of this study. The three types of required inputs are described below, and illustrated in Figure 4.1, under required inputs.

To collect data from the pharmacy domain, the work of Motlhabi [5] was especially useful, as described in Section 2.4.4. Motlhabi had already built a sufficiently large collection of SASL videos. His work provides a template for the general layout of the screens. These form the basis of the content of the XML document. Also useful in Motlhabi's work, was the dialogue he designed. There are three parts to this. First, requesting information from the Deaf patient, secondly asking the pharmacist to fill out the prescription, and lastly presenting that prescription to the Deaf patient. The key requirement discovered from investigating this domain is that it is heavily dependent on non-repetitive inputs.

On examination of the ICDL domain, it was established that the XML has to hold and display only sequential information. The ICDL information proved to be quite extensive, but linear in format. Little user input is required, but unlike the case of the pharmacist domain, there is a requirement to hold occasional images. The key requirements from this domain are that inputs are not required and that the data was purely sequential.

The third arena of required inputs involves viewing the video footage from Motlhabi's work to find other information that can be accommodated such as other dialogues or interactions.

All the information is captured as a set of tags and values in one XML document. This is XML Version 1 and will be the basis for further refinement of the XML.

4.3.1.2 Stage 2

In Stage 2, the work of the previous stage is further refined as illustrated in Figure 4.1. The objective of this stage is to reduce redundancies and remove any incompatible formats held in one XML. The one large XML document is first categorized into User-inputs and Display. All the documents under Display have an identical structure, which can then be parsed identically. All documents under User-inputs are identical in structure, but are able to accommodate a range of predetermined inputs. As in the case of the display documents, the structure of the input documents is identical throughout. The reason for storing the display and input formats in separate documents, is to ensure that the parsing of information from either XML is as simple and repeatable as possible. Creating simpler XMLs has the secondary benefit of increased readability, and also reduces the complexity of work required from the Authoring Tool.

4.3.1.3 Stage 3

In this, the final stage, a means to link the display and inputs is developed, so that the appropriate options can be shown to the users when they are prompted for input. To carry out this stage, modification of the structure of the Display XML was necessary to accommodate an "input" tag. The value of this tag is limited to one of the names of the input XML files (e.g., Diseases.xml). The input XML file then contains the options available to the user for selection. In the case of Diseases.xml, it contains a list of all the available diseases that can be accommodated by SignSupport. It is important to note that all the navigation through the application is sequential, according to the order in the Display XML. Stage 3 provides the user with information on user selection options only when an input is required.

This stage also necessitates the careful, coordinated storage of the resources on the phone. All SignSupport resources fall into a local folder at the root of the phone's local storage, called SignSupport. All Input XMLs are contained in their own folder, as are the display XMLs. Domain specific content such as videos and images are placed in domain-specific folders. The entire file structure is presented in the results section of this Dissertation (Chapter 5).

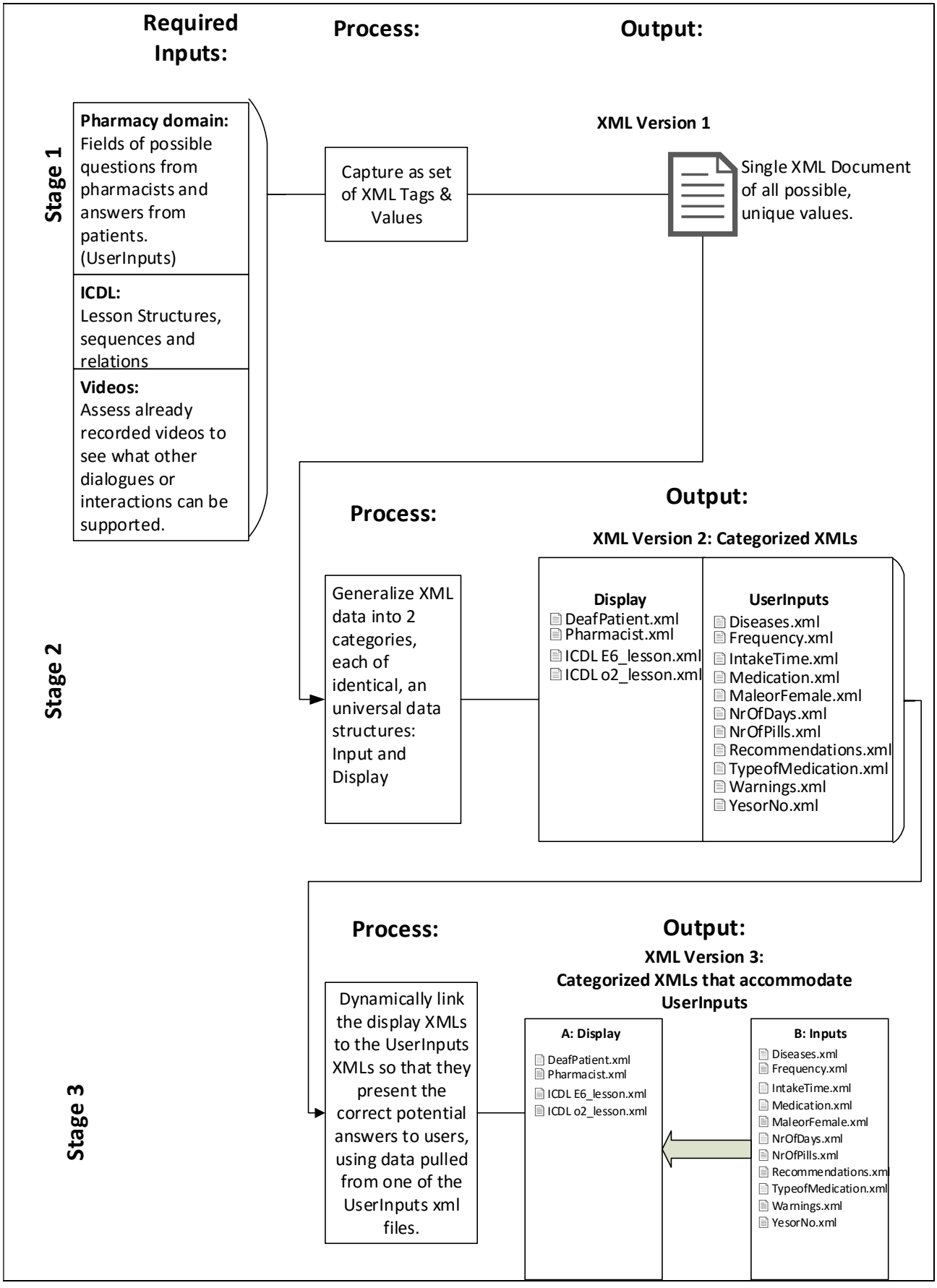

Figure 4.1: Stage 1-3 of XML Data Structure Design

4.3.2 Development of the XML Parser

At the end of the XML data structure design stage, requirements are established for an Android application that will parse two types of XML document to display information and gather inputs. XML parsers are introduced in the previous chapter, Section 3.4.1. The following section aims to fulfil the second objective of this Dissertation or more specifically, speed.

The purpose of the display parser is to read through the display XML, and store the root items “screen” in a list, with each screen having a value for “video_frame”, “video_caption”, “image” and “input”. A Screen object is created in Java with the necessary attributes to hold these values. The values of each asset are stored in the XML as the file location of that asset on the mobile device’s local storage. These values are pulled from the XML text and stored as a string.

The inputs parser is designed to search through a Folder of XML documents first to find the referenced range of inputs stated in the Display XML. Then to parse the document holding those inputs into a drop-down menu, so as to display the necessary choices for the user.

4.3.3 Comparing XML Parsers for Android

The XML parser was tested for speed. The main reason for this is that the parser runs during the “on create ()” methods of each Android activity (as soon as the activity is created). At this point, if the parser takes too long to read through the XMLs and capture all the information, it would delay the display of the next Activity. Timing each parser in this same method, provided comparable results without considering the actual performance of the device. The Android App store requirements and Developer Library state that if an activity is not launched within 15 seconds, the OS will terminate the application automatically. It goes on to say that 200ms is the threshold beyond which users will perceive slowness in an application. SignDIn is designed to fulfil these requirements.

The test for choosing the best XML parser involved the evaluation of three different types of XML Parsing; SAX Parsing, DOM Parsing (Using the third party framework, Java Document Object Model (JDOM)) and Android’s native XML Pull Parsing.

A new mobile application illustrated in Figure 4.3, called the XMLParserTester is built for the purposes of testing the XML parsers. Testing is completed on the same device for which SignDIn is developed, the Samsung Galaxy SIII (described in the beginning of this chapter). The main screen contains three buttons only, each programmed to use a specific type of parser to parse an XML Document closely resembling the display XMLs created in the previous section. The display XML is chosen because it holds significantly more information in different sizes and complexities than the input XMLs. The display XMLs contain strings pointing to assets in the Android file structure, and also contain tags at different levels in the XML Structure. The input XMLs contain strings only, all at one level below the root. The inputs can also vary from holding values of a few characters, to holding values of entire sentences.

The display XML is evaluated in four different sizes. One contains one screen only, taking approximately 10 lines in XML code, another containing 10 screens, then 100 screens, then 1 000 screens. This is done to establish how much information the parser can parse and how long it takes to do this.

To ensure there are no anomalies and that parsing is consistent, each parser runs in a loop and parses the entire XML document a thousand times. A logger class is added to the application to capture the

time when the parser commences and ends each cycle, calculating the time difference in milliseconds to an accuracy of 0.001 seconds. The logger saves this data to a CSV file, which is subsequently analysed to answer two questions:

1. How many (milli-) seconds does it take to parse a Screen object from each XML document?
2. How many screens can be parsed per second, per XML document?

These questions help establish which parser is best suited for the SignDIn application.

Figure 4.2: The XMLParserTester Welcome Screen

4.4 Conclusion

This chapter breaks SignDIn's mobile application down into three parts and outlines the approach taken for each one individually. To achieve the first objective of generalisability, the strategy in designing the XML data structure is described in three stages. Information is gathered from all the available domains, then broken down into common elements, then differentiated into inputs and display, and linked accordingly. The second objective is achieved by building a parser testing application and testing each parser for speed. In doing so, this chapter outlines how the XML parser is selected, evaluated and tested for use in the SignDIn mobile application.

The next chapter presents the results of this work and the output from the design process.

Chapter 5: Results

This chapter presents and discusses the results and outcomes of the methodology described in the previous chapter (Chapter 4). The chapter presents the details of the actual content of the XML data structure and the folder structure of the assets stored locally on the Android device. It goes on to discuss the architecture of the Android application and its parser. Lastly, this chapter will set out the results collated in testing the different types of XML parser, and indicate which parser is used for SignDln.

5.1 The XML Data Structure

The following section describes the resulting XML data structure created from Section 4.3.1 of the methodology. This section addresses the first and second research objectives. It starts with an explanation of how the assets (videos, images and XML documents) are stored on the device, then gives descriptions of the XML documents filtering and presenting those assets.

5.1.1 Storage of Assets

As mentioned earlier, all SignSupport assets are stored locally on the mobile device. These assets include SASL videos, images and XML documents. The following section gives a brief description of the folder structure holding these assets. The SignSupport folder is the root of this file system. Each domain has a dedicated folder. Figure 5.1 shows how the information for the ICDL and pharmacy domains are held in their respective folders. Each domain folder holds domain-specific data separated according to the format. In addition, three folders are domain independent; the StarterXML folder, the Inputs folder, and User Data folder. The StarterXML is so called because it is the first folder parsed for content. These are the options available to the user in the first screen seen by the user. The inputs folder holds all the possible ranges of inputs, each in individual XML documents. This is to separate the input functionality from what is displayed. Finally, the user data folder holds the user information captured from the application in a text file.

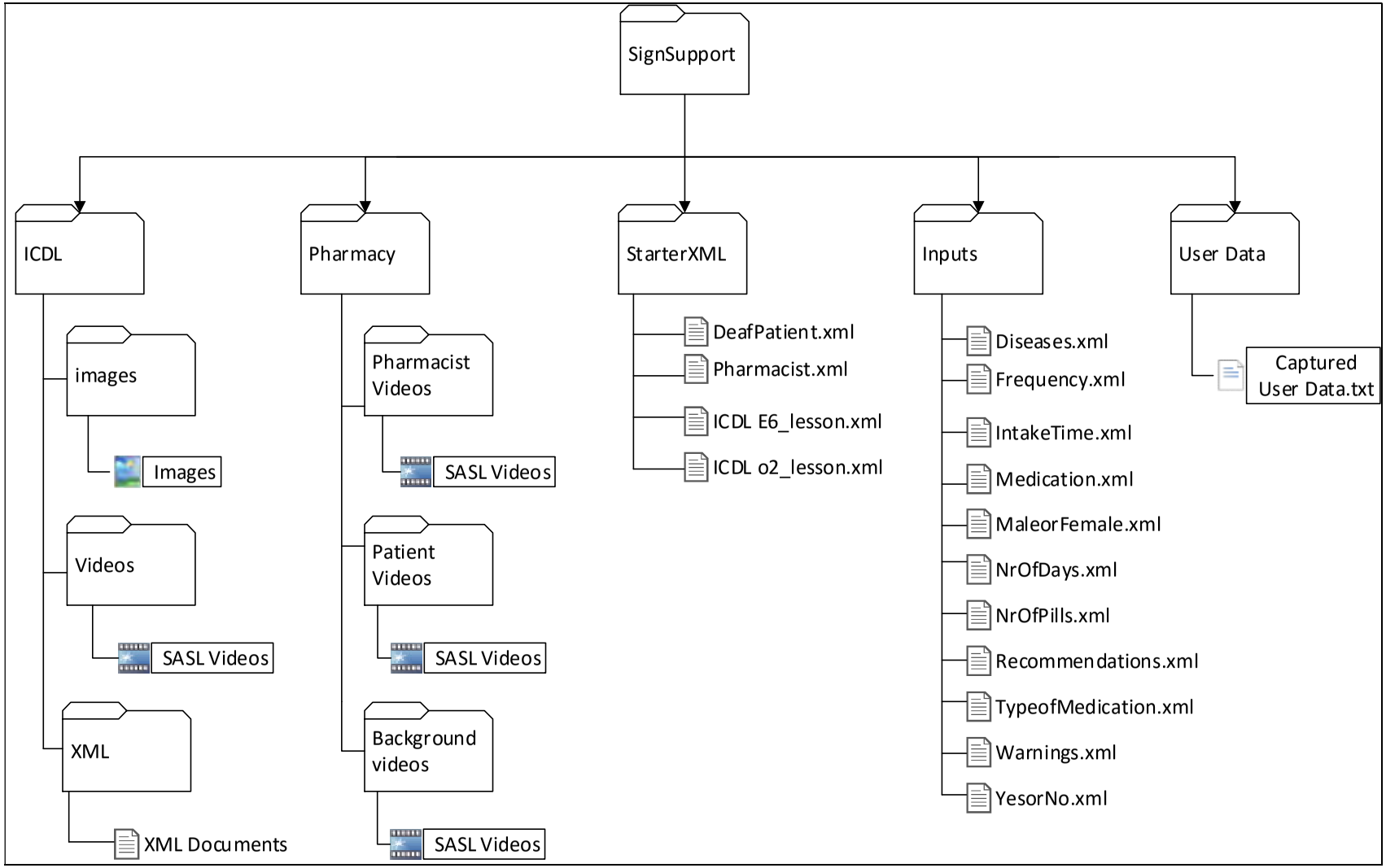

Figure 5.1: How SignSupport Stores its Assets

5.1.2 The Display XML documents

Based on the outputs of stage 3 of the design process (Section 4.3.1.3), it is established that each screen displayed to the user is represented in an XML file (Appendix A) in a <screen> tag (Figure 5.2 a and b). The layout of the actual screen is based on Motlhabi's [5] work. The screens have a next and back button, and each screen shows a unique title or screen ID, a video, a video caption and an image resource where necessary. These screens are represented sequentially in the XML document. The <screenID> tags contain the unique id of the screen. The <video_frame> tag contains the file path to the actual video. The <video_caption> contains a simple text string that is displayed on the screen. These three tags are all used in both domains. The <image> tag is similar to the video tag, in that it maps a file path to the image in question. This is optional, with a default image, as shown in the one taken in the screenshot in Figure 5.2. Lastly, the input tag is the name of the XML file holding the values available as options for the user when selecting the input drop-down menu. These values are parsed by an XML parser, and displayed in the drop-down menu. For many of the ICDL screens, no input is taken from the user, as shown in B of Figure 5.2.

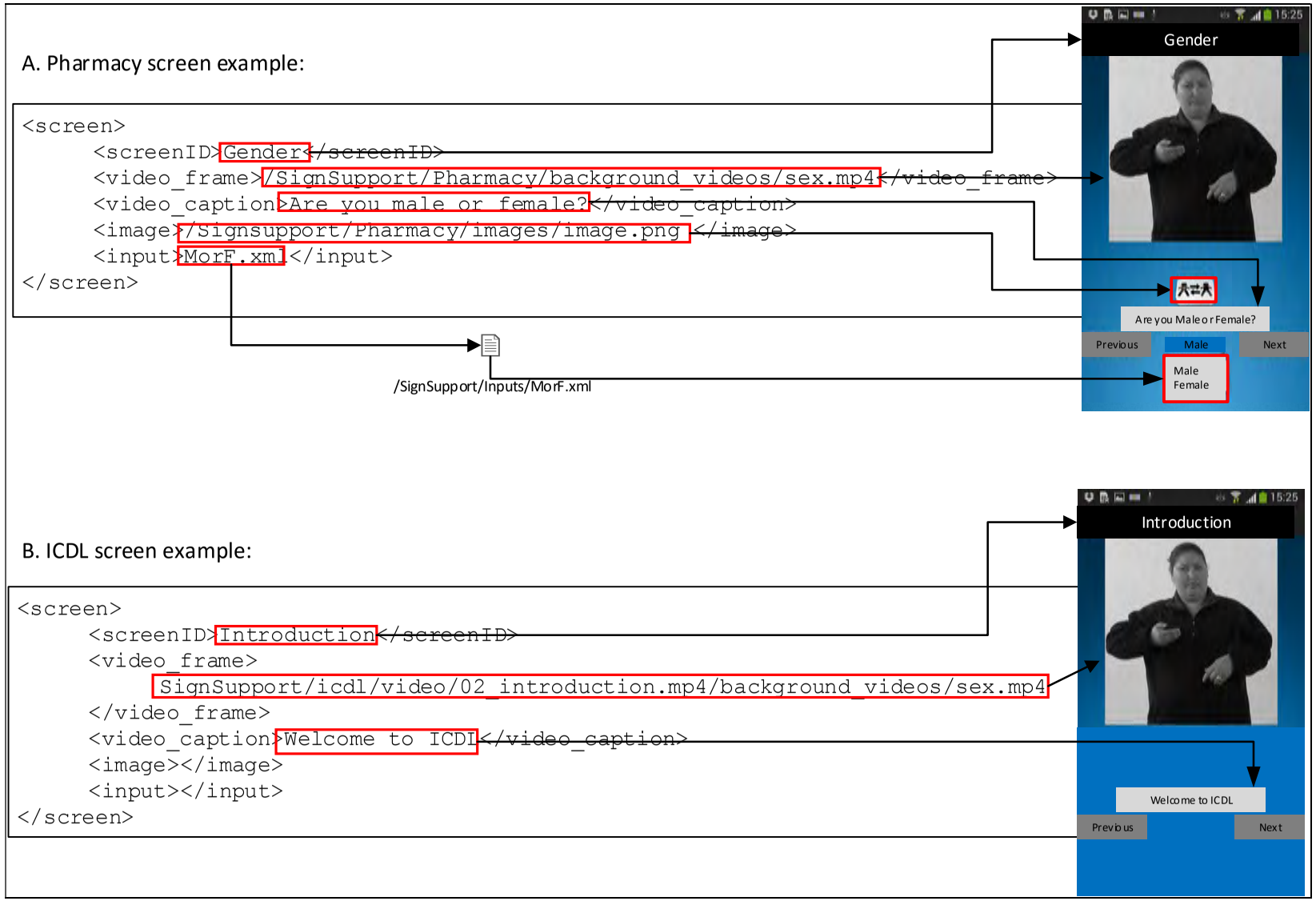

Figure 5.2: Display XML of (a) a Pharmacy screen and (b) an ICDL screen

5.1.3 The Input XML Documents

Figure 5.3 shows one of the User Input XML documents developed in stage 3 of the design. This one gives the option of gender choice. Another example is provided in Appendix B, that of the list of available diseases. At the time of the data structure design, there was no requirement for the ICDL screens to take input. As such, the input tag can be populated optionally. When a content creator is building a screen for display, and wants to prompt the user for a response to his/her gender, the name of this XML file (gender.xml) is included as a value under the `<input>` tag. The parser built into SignDIn then locates the file with that name (in the inputs folder), parses it and displays the appropriate options to the user in a drop down menu (Figure 5.4). Furthermore, the inputs are not parsed from a single, fixed, XML document, but from a folder holding these XML documents. The parser automatically knows where to find the input XML documents, and merely needs to be told which document to reference. If a content author wishes to create a new type of input, he/she must save the range of choices that populate the dropdown menu in a XML document, give it a unique title and store it in in the inputs folder locally on the Mobile Device.

The following section presents the final, comprehensive representation of the SignDIn application to show how the XML parsers are connected to the Android Activities and the SignSupport XML documents and assets.

```
<items>
      <item>Male</item>
      <item>Female</item>
</items>
```

Figure 5.3: Gender.xml as an example of an input XML

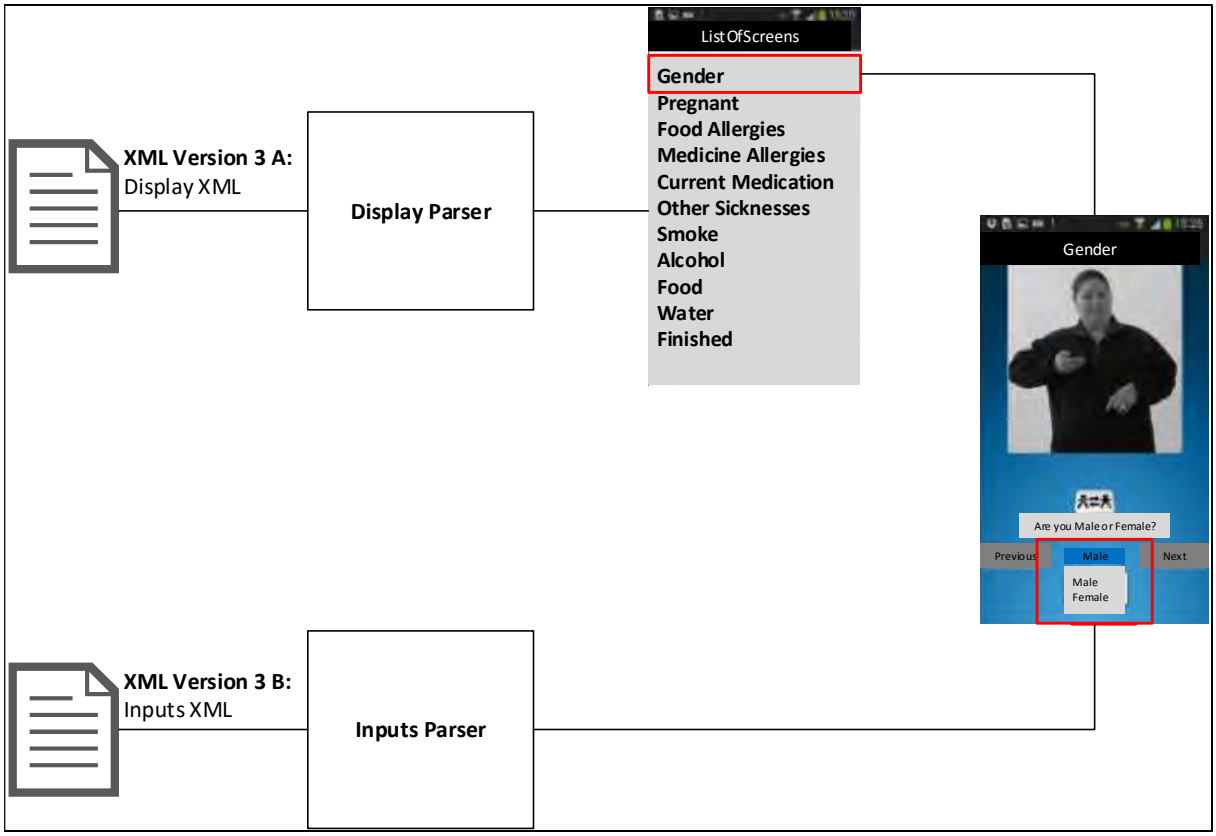

Figure 5.4: The Inputs Parser - Refer also to Figure 4.1

5.1.4 An Example of SignDIn Applied to Two Different domains.

This section presents an example of how the XML data structure can be applied to two different domains, specifically, the ICDL and pharmacy domains. Figure 5.5 below gives a brief example of an XML data structure holding information for the ICDL domain. Here, the screens appear in the same sequence in which they are represented in the XML data structure. For the purposes of this example, there are only three screens: “Introduction”, “Examples” and “Task Description”. None of the screens use an image resource, so it is not necessary to populate the input tag. In this case, the screen is populated with a default image, or none at all. Figure 5.6 presents the XML data structure for the pharmacy domain. When parsed by the XML parser, it displays three videos, in the sequence of the XML data structure. The main difference between the pharmacy and ICDL Data content is that the pharmacy domain requires user input. In this case, the patient is prompted for “*Gender*”, “*Pregnancy*” and “*Allergy*”. This is why the input tags are populated here. These tags are populated with the filenames of the input XMLs (see figure 5.4). These XMLs hold a simple list of options that are presented to the patient.

```
<?xml version="1.0" encoding="UTF-8"?>
<screens>
      <screen>
            <screenID>Introduction</screenID>
            <video_frame>
                  SignSupport/icdl/video/O2_introduction.mp4
            </video_frame>
            <video_caption>Introduction</video_caption>
            <image></image>
            <input></input>
      </screen>

      <screen>
            <screenID>Examples</screenID>
            <video_frame>
                  SignSupport/icdl/video/O2_examples.mp4
            </video_frame>
            <video_caption>Examples</video_caption>
            <image></image>
            <input></input>
      </screen>

      <screen>
            <screenID>Task Description</screenID>
            <video_frame>
                  SignSupport/icdl/video/O2_task_description.mp4
            </video_frame>
            <video_caption>Task Description</video_caption>
            <image></image>
            <input></input>
      </screen>
</screens>
```

Figure 5.5: The ICDL Domain stored in SignDIn's XML Data Structure

```
<?xml version="YorN.xml.0"?>
<screens>
      <screen>
            <screenID>Gender</screenID>
            <video_frame>
                  /SignSupport/Pharmacy/background_videos/sex.mp4
            </video_frame>
            <video_caption>Are you male or female?</video_caption>
            <image>/Signsupport/Pharmacy/images/image.png</image>
            <input>MorF.xml</input>
      </screen>

      <screen>
            <screenID>Pregnant</screenID>
            <video_frame>
                  /SignSupport/Pharmacy/background_videos/pregnant.mp4
            </video_frame>
            <video_caption>Are you Pregnant?</video_caption>
            <image></image>
            <input>YorN.xml</input>
      </screen>

      <screen>
            <screenID>Food Allergies</screenID>
            <video_frame>
                  /SignSupport/Pharmacy/background_videos/
                  food_allergy.mp4
            </video_frame>
            <video_caption>
                  Do you have a food allergy?
            </video_caption>
            <image>/Signsupport/Pharmacy/images/image.png</image>
            <input>YorN.xml</input>
      </screen>
</screens>
```

Figure 5.6: The Pharmacy domain stored in SignDIn's XML Data Structure

5.2 SignDIn and the XML Parser

Figure 5.5 is an illustration of the completed design of the SignDIn application. As such, almost all the information provided for the user is pulled dynamically from the XML documents. There are two parts to the application that are fixed in all domains – the welcome screen video and the "*Next*" and "*Previous"* buttons on the Screen Detail Activity. It is anticipated that the welcome video will provide a short welcome, and prompt the user to select one of the items in the list below that. Each list item in the welcome screen (A in Figure 5.5) is placed dynamically when the application creates this first screen and looks at the StarterXML folder to see what needs to be displayed.

When a user selects an item on this list, the next screen is called the List Activity (e.g. *DeafPatient.xml* in B of Figure 5.5). As this screen is created, it calls up the XML pull parser and parses the relevant XML document, to display each Screen as an item in a list. Remember, that "Screen" is its own class and each screen is stored as a Screen object, with a value for Screen ID, Video_frame, Video_caption, Image and input. These items are all stored in memory as Screen objects.

When a user selects an item in this List Activity, a new Activity is created, the Screen Detail Activity (C in Figure 5). This activity pulls together all the values for this particular screen and displays all information. All browsing through the Display XMLs is done in this one activity. If any of the fields in the XML are left blank, the value is ignored. If any of the fields point to a non-existent file, the user is alerted by a pop up prompt called a "Toast", but the application continues to work anyway. The application uses this one activity to perform all the video viewing and inputs. This is the core of the application, and the user is allowed to browse back and forth with the permanent "previous" and "next" keys permanently onscreen. These keys are built into the application and are not pulled from the XML documents. Using the recursive arrow on the phone will take the user up one level, back to the previous activities, until he/she returns to the first activity, and eventually exits from the application.

The input options on each Screen are displayed in a drop-down menu called a Spinner. A spinner can take a list of objects, in this case, strings, and display them for the user to select. This list of objects is pulled from a List, populated from a second parser called the "*Items Pull Parser*" which looks at the inputs folder and, depending on the value held in the Display XML under `<input>`, finds the XML document of the same name and parses its content into the list feeding the Spinner.

Each time the user makes a selection in the Spinner, the application records his/her selection in a text file held in the folder *"/SignSupport/User Data*". Once the user has come to the end of the list of Screens, he/she is presented with a new Activity: the Results Activity. This activity shows the results just captured, and gives him/her the option to review previous text files that have been captured. These user data-files are persistent, and are saved with a time stamp. This makes them useful for further study and analysis; say for example if SignDIn is used to collect surveys from Deaf people.

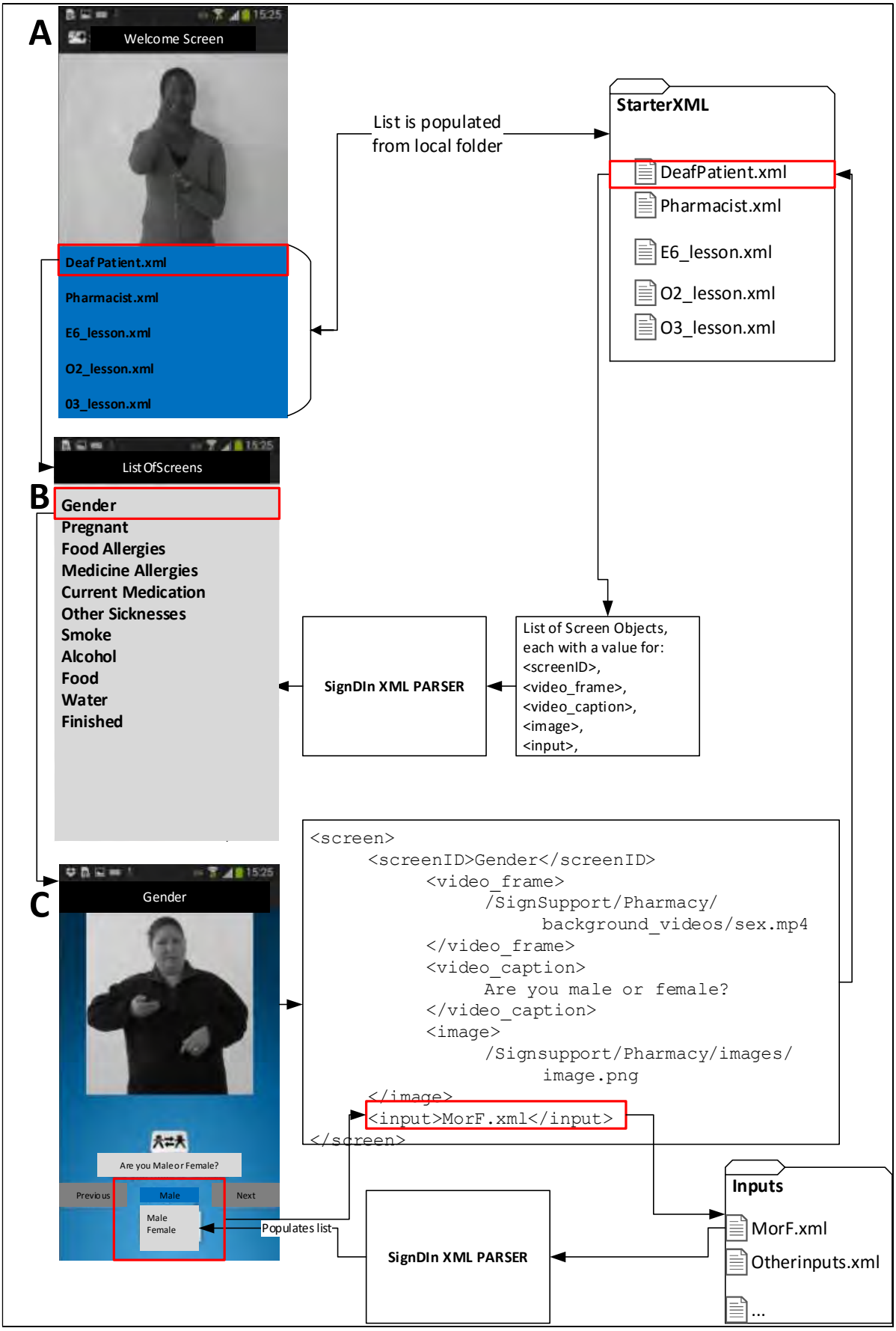

Figure 5.5: The Resulting Design of the SignDIn Application

5.3 Comparing XML Parsers for Android

This section presents the results of the XML parser tests carried out in Section 4.3.3 of the methodology and addresses the second objective of this Dissertation, which is to assess the speed of the application.

5.3.1 Time per Screen Object

The first metric to be analysed is the time taken to parse the entire XML document. This parser runs in the `OnCreate()` method when an activity is started, and will give us an idea of the delays between activities.

From Figure 5.6, it is evident that the larger the XML document, the longer it takes to parse. It is also clear that the native XML parser, the XML pull parser, is the fastest in all circumstances. Appendix C shows the consistency of parsing times through the 1 000 cycle test (See Appendix C, Table C.1 for the raw data). It is also interesting to know the differences for each size of XML document. When you compare parsing an XML document of one screen versus one of a thousand screens, the pull parser is about 175 times slower for the larger XML document. If you do the same comparison on the other parsers, the SAX parser is about 340 times slower and the JDOM parser about 400 times slower than the XML pull parser.

It is, however, unrealistic to expect any content creator to build a dialogue of a thousand screens. Ten screens is more likely. In these circumstances, the XML pull parser averages 0.0016 seconds to parse the entire document. Comparing this with the Android library's definition of a "slow" response as being 200 milliseconds (0.2 seconds) this is a difficult metric to comprehend. The next section provides an alternative answer, in terms of how many screens can be parsed in that 0.2 second time limit.

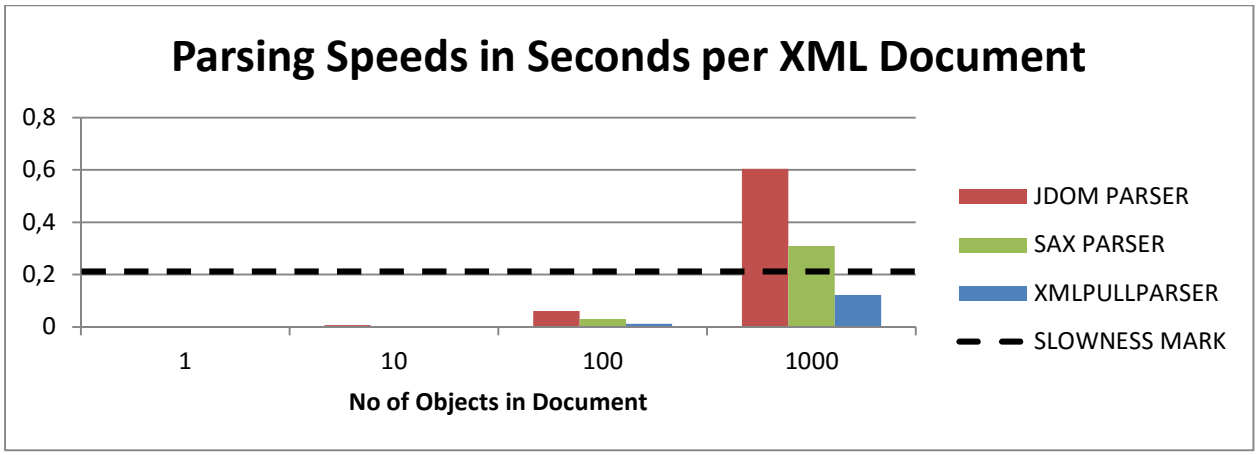

Figure 5.6: Parsing Speeds in average seconds per XML Document over 1000 cycles.

5.3.2 Parsed Screens Throughput

Considering the parser throughput is perhaps more a comparable measure than that presented above. Table 5.2 and Figure 5.9 summarize the Screen Throughput of each parser. It appears that all the parsers are able to parse the well over a thousand Screen objects into memory in under the required 200 milliseconds.

Table 5.1: Parser Throughput (number of Screens parse-able in 0.2 Seconds)

Parser Type:	1 Screen	10 Screens	100 Screens	1000 Screens
XMLPULLPARSER	286	1216	1736	1645
JDOM PARSER	132	290	327	331
SAX PARSER	221	601	646	646

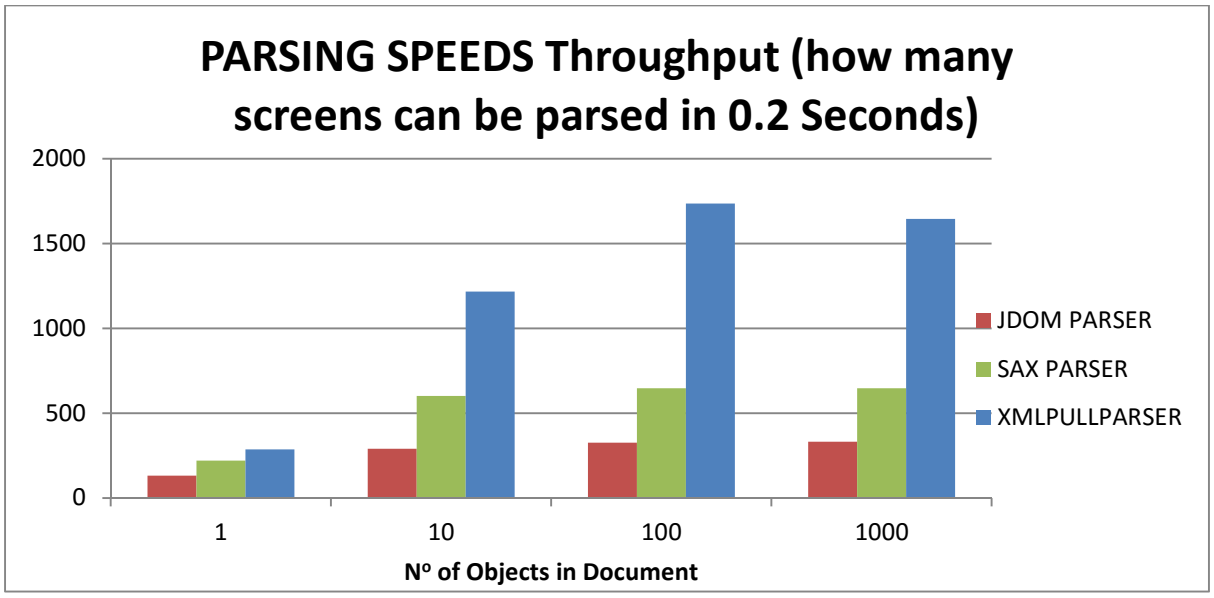

Figure 5.7 Number of Screen Objects parsed in 0.2 Seconds

5.4 Discussion

This Dissertation aims to achieve two primary objectives. The first, achieving generalisability is achieved through the design of the XML data structure. The second objective is achieved through the evaluation of alternative parsers.

The storage of assets in the SignDIn application is designed in such a way as to ensure that the Input and display information are separated logically. In this way, SignDIn will be universal across domains in displaying information, but very unique in the range of inputs a user can give. This also means that input XMLs can be used across domains, where repetition does occur.

The data structure presented in this chapter is suitable for use in SignSupport because it is applicable to multiple domains of use. Examples are provided of the XML data structure holding information relevant to two, completely unrelated, domains of use. Through these examples, it is shown that, without altering the basic structure of the XML, the data structure can be made to accommodate both the ICDL and pharmacy domains. By doing this, the results show that as long as each respective domain can be abstracted into a sequenced set of videos, with optional supporting content (images, text, etc.) then the SignDIn application is able to store this information in the XML data structure as well as parse the information and present it in a uniform way to the user. This satisfies the first objective of this Dissertation, because the XML data structure provides a standard format into which any domain can be broken down, and stored and parsed by SignDIn.

Part of this research is to test the XML Parsers for their speed to ensure that reading information from an XML document does not cause any unacceptable latency in the use of the application. Three parsers are assessed, each offering a unique set of benefits and drawbacks, mostly depending on the structure of the XML being parsed, the size of the XML document, and the frequency of parsing. The parser is initialized when activities are created and when user input is selected from drop-down menus. As such, the parser is tested for speed, as this directly influences the user's perception of slowness. The significantly better performance by Android's XML pull parser makes it the clear parser of choice in building the SignDIn Application. This is the parser that is used and the parser that is recommended for use in future iterations of SignDIn.

The final chapter concludes this dissertation by tying the original objectives together with the outcomes of the research.

Chapter 6: Synthesis and Conclusion

The research aims to build on existing work accomplished by researchers working on the SignSupport platform of mobile applications facilitating communication for Deaf people. The earliest version of SignSupport is merely a proof of concept carried out on a PC [4]. The next version runs on a mobile device completely [14]. This version also introduces the concept of the Authoring Tool, a PC based application facilitating the building of conversations and their relevant SASL videos for loading onto a mobile device. The third version of SignSupport applies this to a single, specific domain of use, namely the pharmacy [17]. The penultimate version of the application is built specifically for the pharmacy domain and is developed for the Android mobile operating system [5]. Researchers have established methods for building dialogues, filming SASL videos and verifying those SASL videos. Parallel to this study, the ICDL domain was added to SignSupport. What remains to be overcome, is to build a mobile application that can easily be extended to new domains. This can be done through a mechanism facilitating the transfer of information from the Authoring Tool to the mobile device. This increases access to the mobile-assisted translation of SASL with the intention of providing a solution for assisting Deaf people to function in a predominantly hearing society. This mobile application and the XML data structure upon which it relies is called SignDIn, and is the result of the work done in this Dissertation.

The objective of this chapter is to provide a synthesis of the main findings emerging from the research, and to discuss their implications. The limitations of the research are outlined and recommendations for future studies noted.

6.1 Key Outputs

6.1.1 Generalisability

The first objective of this research aims to develop a data structure in a generalizable format that is easily applicable to multiple domains of use. Generalisability, in specific, refers to the ability to apply a data structure broadly to other domains of use. Before this can be accomplished, it is necessary to establish the technology to be used to build a data structure for SignDIn. Chapter 3 addresses this preliminary problem by assessing three candidates, namely, Relational Databases, JSON and XML. XML is assessed (Section 3.3) as being the most appropriate data structure. At this stage, XML Parsing is introduced (Section 3.4), as this is the mechanism through which information will be read from the data structure and displayed to the Deaf user.

Following this, the logical structure of the XML data structure is designed. This process (described in Section 4.3.1) involves three separate stages, each stage refining the information composed in the previous one and building on it in order to produce the most suitable data structure. The result of the final stage is two types of XML documents: Display XMLs (Section 5.2.2) to facilitate the display of screens and Input XMLs (Section 5.2.3) to show the user the options available when prompted for input. Generalisability is achieved by storing information about each domain in a simple format in XML. Every screen displayed to the user is broken down into video, caption, image and input. This method of storing information is domain independent and can be abstracted from a dialogue in any domain. The result of this level of generalisability is that SignDIn is context and Sign Language independent.

6.1.2 Parser Efficiency

SignDIn is assessed for its performance in parsing the information held in the XML documents. The architecture of the application requires the XML document, generally, to be parsed during screen changes. It is established that a delay of more than 0.2 seconds between screen changes will give the user the impression that the application is not responding (Section 5.3). Using a completely separate mobile application designed only for evaluating parsers, three different XML parsers are evaluated to find the fastest one (Section 5.4). These results show how much information can potentially be parsed without the user noticing a delay. The parsing speed (Figure 5.8) and data throughput (Figure 5.9) is deemed acceptable to accommodate SignDIn and indicates that it will not pose any realistic problem for users or content builders.

6.2 Evaluation of the Research Design and Methodology

6.2.1 Data Structure Limitations

Firstly, selecting the technology of the data structure used was based on a recognisably subjective assessment of two technologies (Section 3.3). Secondly, the design of the current data structure limits the usefulness of this work to cases where all the content is stored locally on the mobile device only. When content is being rapidly developed, it is better to store all this information in a Relational Database, as databases like these are capable of handling rapid changes to the data. If the database is stored on a networked location, the use of XML may still be necessary, as Relational Database tables are typically not updated directly through Internet protocols, but rather through formats conducive to transferring information over the Internet, such as XML. This would necessitate the redevelopment of the XML parser. However, the current methodology adopted by Motlhabi [5] for building content for SignSupport is a slow process. A dialogue must be built, then translated and then filmed in SASL. Next the content must be checked for inaccuracies before eventually being transferred to the mobile device. If this process is improved to the extent that new content is constantly being built for SignSupport, then a Relational Database (RDB) will better serve these needs.

The general outlay of the screen, as shown in Figure 5.2, is based on the outlay of the screens from Motlhabi's study. While these fields are optional, the basic premise of a back/forward button, and the sequential stepping through of a conversation cannot be altered when using the current data structure.

The close-ended conversation style is used as the template in designing the data structure. Both domains used in this study (ICDL and Pharmacy) use a close-ended conversation structure. Because of this, SignDIn can only support close-ended conversation scenarios. This is acknowledged as a limitation of this work.

6.3 Future Directions

6.3.1 User Testing

SignDIn is designed purely to fulfil the technical requirement of providing content to a mobile application. The resulting mobile application is tested only against quantifiable measures of speed. A further challenge remains to test the application with Deaf test subjects. Such an experiment will provide more useful information if the anticipated authoring tool is tested concurrently with a domain specialist, thus providing insight into the entire process of loading content onto the mobile device

(Figure 1.1) and viewing that content. Such an experiment will build on the work done thus far by providing user-focused inputs which can be used to improve the SignSupport application.

6.3.2 Other mobile platforms

There is also the potential to apply the same XML pull parser to other platforms and to carry out inter-platform comparisons of this parser.

6.4 Main Conclusions

SignDIn and its reliant XML data structure fulfils all the research objectives set out in this Dissertation. The design of the SignDln application and the supporting XML data structure has facilitated a means for SignSupport to be applied easily to other domains of use without reprogramming or rewriting the code of the application. In other words, it is generalisable. The Data Structure sets out a framework for output from the desktop-based authoring tool, which is introduced in Section 1.1. This facilitates an application which is generalisable to other domains of use, and in essence, future-proofed. With the authoring tool, the SignSupport system will have full generalisability, facilitated and dictated by the XML data structure, and executed by the SignDln mobile application. The mobile application is built with the fastest XML parser of those assessed, ensuring that the SignDIn mobile application is responsive when parsing the XML data structure.

While acknowledging some limitations, this research offers the facility to add new content to SignSupport without building a new mobile application. By gathering and building new SASL content, SignDIn can be used to offer a means for Deaf people to cope in more environments where dialogue can be abstracted in the appropriate way.

Overall, this Dissertation has built the foundation upon which SignSupport can be broadened to new domains and extended within existing domains, ensuring continued work with DCCT.

Bibliography

[1] *www.SignSupport.org*. Available: http://www.signsupport.org/about/.

[2] T. Reagan, C. Penn and D. Ogilvy, "From policy to practice: Sign language developments in post-apartheid South Africa," *Language Policy,* vol. 5, pp. 187-208, 2006.

[3] J. Shapiro, *No Pity: People with Disabilities Forging a New Civil Rights Movement.* New York: Times Books, 1993.

[4] K. Looijesteijn, "The design of a Deaf-to-hearing communication aid for South Africa," *Unpublished MSc Thesis, Delft University of Technology, Netherlands,* 2009.

[5] M. T. Motlhabi WD., "Usability and Content Verification of a Mobile Tool to help a Deaf person with Pharmaceutical Instruction," Masters Paper, University of The Western Cape, South Africa, 2013.

[6] E. Blake, W. Tucker and M. Glaser, "Towards communication and information access for Deaf people: research article," South *African Computer Journal,* vol. 54, pp. 10-19, 2014.

[7] C. Padden, "2.4 the Deaf Community and the Culture of Deaf People," Constructing *Deafness,* vol. 40, 1991.

[8] Cavender, A, et al. *Accessible Writing Guide. Special Interest Group on Accessibility in Computing. Retrieved from http://www.sigaccess.org/welcome-to-sigaccess/resources/accessible-writing-guide/,* 2014.

[9] M. Glaser and W. D. Tucker, "Telecommunications bridging between deaf and hearing users in South Africa," in *Conference and Workshop on Assistive Technologies for Vision and Hearing Impairment (CVHI), Granada, Spain,* 2004, .

[10] M. Glaser and T. Lorenzo, "Developing literacy with deaf adults.in Watermeyer, B. et al., eds.," in *Disability and Social Change: A South African Agenda.* Anonymous Cape Town, South Africa: HSRC Press, 2006.

[11] C. Penn and T. Reagan, "The Properties of South African Sign Language: Lexical Diversity & Syntactic Unity," *Sign Language Studies,* vol. 85, pp. 319-327, 1994.

[12] http://www.dcct.org.za/?q=about, *About DCCT*, [Last accessed: March 2014].

[13] M. Glaser and W. D. Tucker, "Telecommunications bridging between deaf and hearing users in South Africa," in *Conference and Workshop on Assistive Technologies for Vision and Hearing Impairment (CVHI), Granada, Spain,* 2004, .

[14] M. Mutemwa and W. D. Tucker, "A mobile deaf-to-hearing communication aid for medical diagnosis," in *Southern African Telecommunication Networks and Applications Conference (SATNAC),* 2010, pp. 379-384.

[15] P. Chininthorn, A. Freudenthal, M. Glaser and W. Tucker, "Mobile communication tools for a South African deaf patient in a pharmacy context,"*Proc.Information Society Technologies-Africa, (IST-Africa 2012), Dar Es Salaam, Tanzania,* 2012.

[16] W. D. Tucker, M. Glaser and J. Lewis, "SoftBridge in action: The first deaf telephony pilot," in *Southern African Telecommunications Networks & Applications Conference (SATNAC),* 2003, pp. 293-294.

[17] P. Chininthorn, "Communication Tool Design from Deaf to Hearing in South African,"2011.

[18] (). *Google announces Android has surpassed 1 billion device activations*.

[19] W. C. Stokoe, "Sign language structure: An outline of the visual communication systems of the American deaf," *Journal of Deaf Studies and Deaf Education,* vol. 10, pp. 3-37, 2005.

[20] P. Miller, "The nature and efficiency of the word reading strategies of orally raised deaf students," *Journal of Deaf Studies and Deaf Education,* vol. 14, pp. 344-361, 2009.

[21]Ngubane, B. S. *For the Launch of the Pilot Project Telephone Interpreting Services for South Africa (TISSA). Katlehong Police Station: Minister of Arts, Culture, Science and Technology. Retrieved from http://www.dacst.gov.za/speeches.minister/mar2002/tissa_launch.htm*. 2002.

[22] Z. P. Jordan, "Address by the Minister of Arts and Culture, Dr Z Pallo Jordan, at the launch of the Telephone Interpreting Service for South Africa (TISSA)." *South African Government Information,* 2005.

[23] K. van der Merwe, "Want access to government services? With TISSA you can!"*SAPS Journal, September,* 2005.

[24] A. Cavender, R. E. Ladner and E. A. Riskin, "MobileASL: Intelligibility of sign language video as constrained by mobile phone technology," in *Proceedings of the 8th International ACM SIGACCESS Conference on Computers and Accessibility,* 2006, pp. 71-78.

[25] J. Kim, J. J. Tran, T. W. Johnson, R. Ladner, E. Riskin and J. O. Wobbrock, "Effect of MobileASL on communication among deaf users," in *CHI'11 Extended Abstracts on Human Factors in Computing Systems,* 2011, pp. 2185-2190.

[26] C. Jayant, C. Acuario, W. Johnson, J. Hollier and R. Ladner, "V-braille: Haptic braille perception using a touch-screen and vibration on mobile phones," in *Proceedings of the 12th International ACM SIGACCESS Conference on Computers and Accessibility,* 2010, pp. 295-296.

[27] S. Azenkot, S. Prasain, A. Borning, E. Fortuna, R. E. Ladner and J. O. Wobbrock, "Enhancing independence and safety for blind and deaf-blind public transit riders," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems,* 2011, pp. 3247-3256.

[28] B. M. Jones and N. Johnson, "Assessing the effects of mobile OS design on single-step navigation and task performance," in *Human Interface and the Management of Information. Information and Interaction for Health, Safety, Mobility and Complex Environments* Anonymous Springer, 2013, pp. 383-390.

[29] P. E. Black, *Dictionary of Algorithms and Data Structures.* National Institute of Standards and Technology, 2004.

[30] E. M. Awad, *Systems Analysis and Design.* McGraw-Hill Professional, 1985.

[31] N. Nurseitov, M. Paulson, R. Reynolds and C. Izurieta, "Comparison of JSON and XML Data Interchange Formats: A Case Study." *Caine,* vol. 2009, pp. 157-162, 2009.

[32] M. H. Goadrich and M. P. Rogers, "Smart smartphone development: iOS versus Android," in *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education,* 2011, pp. 607-612.

[33] http://www.saxproject.org/. *SAX*, [Last accessed: July 2014].

[34] http://www.w3.org/tr/dom/, *W3C DOM4 Last Call Working Draft,* [Last accessed: July 2014].

Appendix A: XML v3a – Display Screens – Pharmacy patient example

```
<?xml version="1.0"?>
<screens>
  <screen>
    <screenID>Gender</screenID>

<video_frame>/SignSupport/Pharmacy/background_videos/sex.mp4</video_
frame>
    <video_caption>Are you male or female?</video_caption>
    <image>/Signsupport/Pharmacy/images/image.png</image>
    <input>MorF.xml</input>
  </screen>

   <screen>
    <screenID>Pregnant</screenID>

<video_frame>/SignSupport/Pharmacy/background_videos/pregnant.mp4</v
ideo_frame>
    <video_caption>Are you Pregnant?</video_caption>
    <image>/</image>
    <input>YorN.xml</input>
  </screen>

   <screen>
    <screenID>Food Allergies</screenID>

<video_frame>/SignSupport/Pharmacy/background_videos/food_allegy.mp4
</video_frame>
    <video_caption>Do you have a food allergy?</video_caption>
    <image>/Signsupport/Pharmacy/images/image.png</image>
```

```
    <input>YorN.xml</input>
  </screen>

   <screen>
    <screenID>Medicine Allergies</screenID>

<video_frame>/SignSupport/Pharmacy/background_videos/med_allergy.mp4
</video_frame>
    <video_caption>Do you have a medicine allergy?</video_caption>
    <image>/Signsupport/Pharmacy/images/image.png</image>
    <input>YorN.xml</input>
  </screen>

   <screen>
    <screenID>Current Medication</screenID>

<video_frame>/SignSupport/Pharmacy/background_videos/curr_meds.mp4</
video_frame>
    <video_caption>Are you on any current
medication?</video_caption>
    <image>/Signsupport/Pharmacy/images/image.png</image>
    <input>YorN.xml</input>
  </screen>

   <screen>
    <screenID>Other Sicknesses</screenID>

<video_frame>/SignSupport/Pharmacy/background_videos/med_allergy.mp4
</video_frame>
    <video_caption>Do you have any other sicknesses?</video_caption>
    <image>/Signsupport/Pharmacy/images/image.png</image>
    <input>YorN.xml</input>
  </screen>

   <screen>
    <screenID>Smoke</screenID>

<video_frame>/SignSupport/Pharmacy/background_videos/smoke.mp4</vide
o_frame>
    <video_caption>Do you smoke?</video_caption>
    <image>/Signsupport/Pharmacy/images/image.png</image>
    <input>YorN.xml</input>
  </screen>

   <screen>
    <screenID>Alcohol</screenID>

<video_frame>/SignSupport/Pharmacy/background_videos/drinker.mp4</vi
deo_frame>
    <video_caption>Do you drink alcohol?</video_caption>
    <image>/Signsupport/Pharmacy/images/image.png</image>
    <input>YorN.xml</input>
  </screen>

   <screen>
```

```
    <screenID>Food</screenID>

<video_frame>/SignSupport/Pharmacy/background_videos/food.mp4</video
_frame>
    <video_caption>Do you eat 3 meals a day?</video_caption>
    <image>/Signsupport/Pharmacy/images/image.png</image>
    <input>YorN.xml</input>
  </screen>

   <screen>
    <screenID>Water</screenID>

<video_frame>/SignSupport/Pharmacy/background_videos/water.mp4</vide
o_frame>
    <video_caption>Do you have access to drinkable
water?</video_caption>
    <image>/Signsupport/Pharmacy/images/image.png</image>
    <input>YorN.xml</input>
  </screen>

   <screen>
    <screenID>Finished</screenID>

<video_frame>/SignSupport/Pharmacy/background_videos/end.mp4</video_
frame>
    <video_caption>Thank you, you are done. Please press next to see
your results.</video_caption>
    <image>/Signsupport/Pharmacy/images/image.png</image>
    <input></input>
  </screen>

</screens>
```

Appendix B: XML v3b – Input Options example - diseases

```
<?xml version="1.0"?>
<items>

    <item >Toothache </item>
    <item></item>
    <item >Stroke</item>
    <item >Spinal Injury</item>
    <item >Sinus</item>
    <item >Shingles</item>
    <item >Sexually transmitted disease STI</item>
    <item >Ringworm</item>
    <item >Respiratory Infection</item>
    <item >Pulmonary tuberculosis TB</item>
    <item >Piles</item>
    <item >Mumps</item>
    <item >Meningitis</item>
    <item >Lice</item>
    <item >Jaundice</item>
    <item >Insomnia</item>
    <item >Influenza</item>
    <item >Indigestion</item>
    <item >HIV</item>
    <item >Hepatitis B</item>
    <item >Hepatitis</item>
    <item >Heartburn</item>
    <item >Heart Disease</item>
    <item >Gout</item>
    <item >Food Poisoning</item>
    <item >Fever</item>
    <item >Eczema</item>
    <item >Diarrhea</item>
    <item >Diabetes</item>
    <item >Dementia</item>
    <item >Dehydration</item>
    <item >Dandruff</item>
    <item >Constipation</item>
    <item >Chickenpox</item>
    <item >Cellulitis</item>
    <item >Cancer</item>
    <item >Burns</item>
    <item >Bulimia</item>
    <item >Bipolar</item>
    <item >Arthritis</item>
    <item >Anxiety</item>
    <item >Anorexia</item>
    <item >Anaemia</item>
    <item >Ulcer</item>
    <item >Alcohol Poisoning</item>
    <item >Acne</item>

</items>
```

Appendix C: Results from 1000 Cycle Parser Test

Table C.2: Parsing Speeds in Average Seconds per XML Document

Parser Type:	1 Screen Ave	10 Screens Ave	100 Screens Ave	1000 Screens Ave
XMLPULLPARSER	0.000698699	0.001644645	0.011520521	0.121603604
JDOM PARSER	0.001515516	0.006896897	0.061236236	0.60381982
SAX PARSER	0.000903904	0.003325325	0.03094995	0.309600601

JDOM Parser

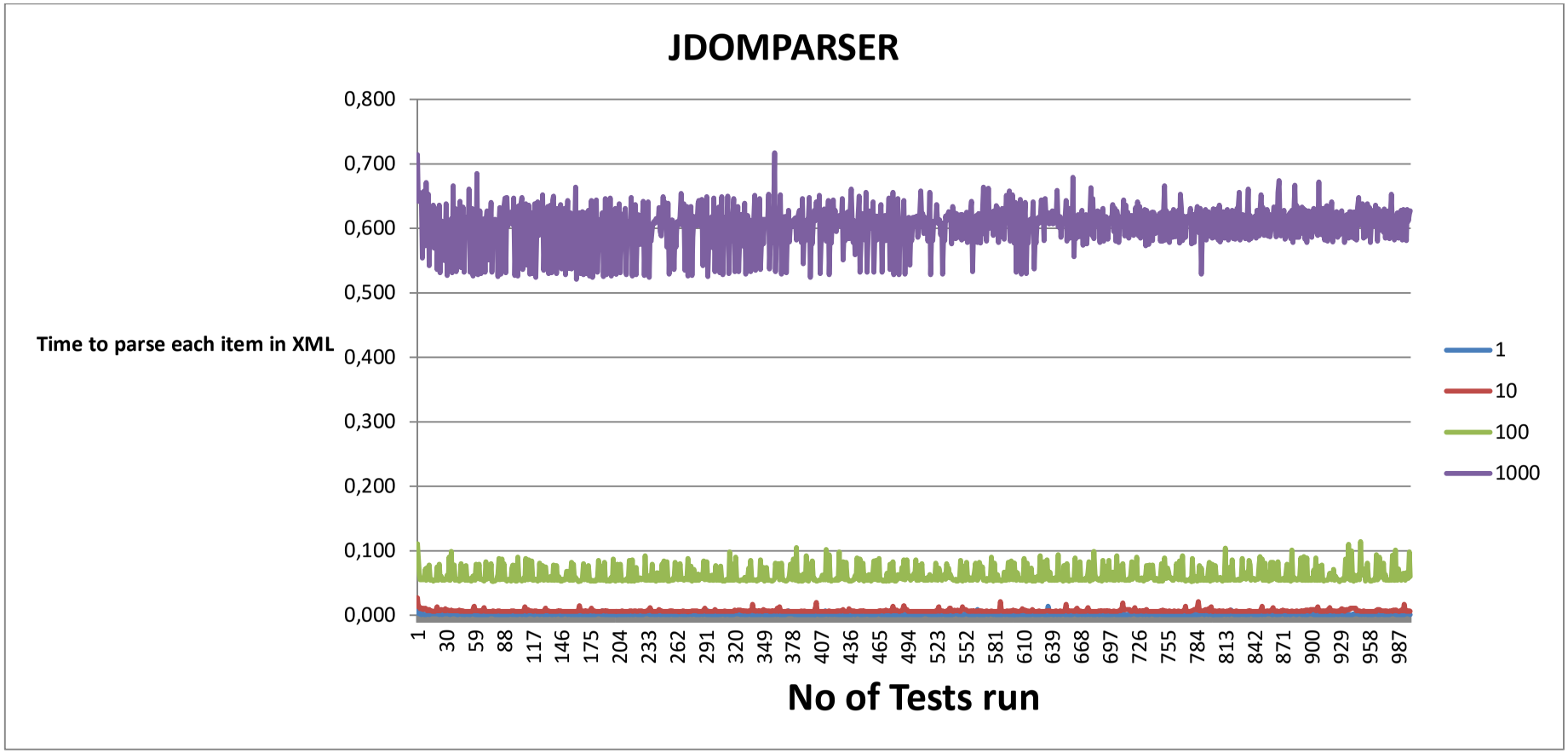

Figure C.1: Variance in time to parse a screen object in seconds for the JDOM Parser

SAX Parsers

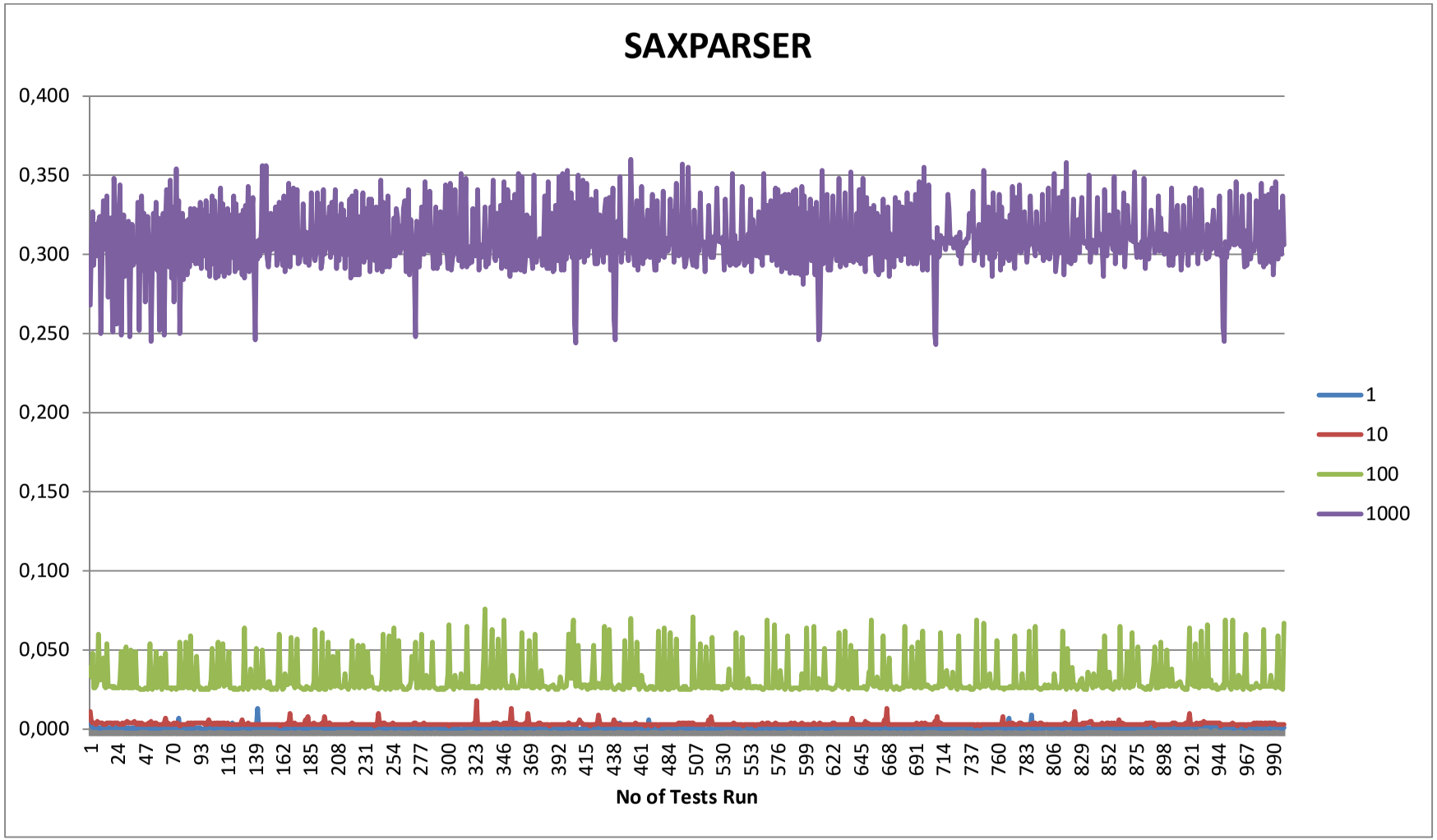

Figure C.2: Variance in time to parse a screen object in seconds for the SAX Parser

XML Pull Parser

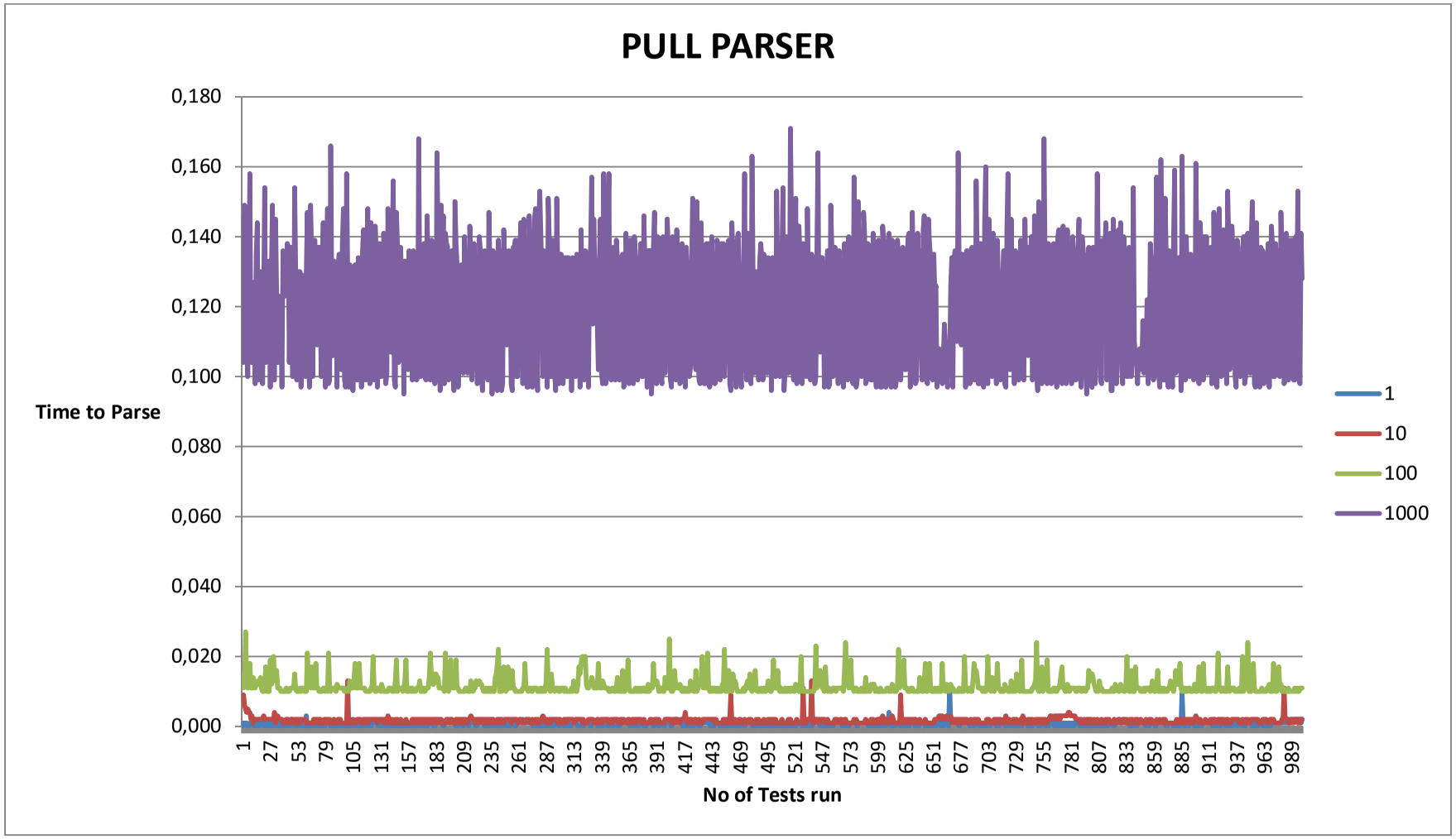

Figure C.3: Variance in time to parse a screen object in seconds for the PULL Parser